AFRL-RI-RS-TR-2015-057

DETAILED PHONETIC LABELING OF MULTI-LANGUAGE DATABASE FOR SPOKEN LANGUAGE PROCESSING APPLICATIONS

BINGHAMTON UNIVERSITY

MARCH 2015

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. This report is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2015-057 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/ S /
STANLEY WENNDT
Work Unit Manager

/ S /
ROBERT KAMINSKI for
WARREN H. DEBANY, JR.
Technical Advisor, Information
Exploitation & Operations Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
MARCH 2015	FINAL TECHNICAL REPORT	JAN 2012 – JAN 2015

4. TITLE AND SUBTITLE

DETAILED PHONETIC LABELING OF MULTI-LANGUAGE DATABASE FOR SPOKEN LANGUAGE PROCESSING APPLICATIONS

5a. CONTRACT NUMBER
FA8750-12-1-0093

5b. GRANT NUMBER
N/A

5c. PROGRAM ELEMENT NUMBER
65502F

6. AUTHOR(S)

Stephen A. Zahorian

5d. PROJECT NUMBER
3480

5e. TASK NUMBER
BI

5f. WORK UNIT NUMBER
NG

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Binghamton University
Department of Electrical and Computer Engineering
Binghamton, NY 13902-6000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/RIGC
525 Brooks Road
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/RI

11. SPONSOR/MONITOR'S REPORT NUMBER
AFRL-RI-RS-TR-2015-057

12. DISTRIBUTION AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The main objective of this research was to explore and refine methods for detailed phonetic labeling (English or Russian) and character level labeling (Mandarin). Much of the work involved new front end signal processing methods designed to improve acoustic phonetic representations for speech. This resulted in recognition rates for TIMIT (English) of 74%, among the highest reported in the literature. A complete character recognition system for Mandarin was developed and tested. Character recognition rates as high as 88% were obtained, using an approximately 40 training databases. For the case of Russian, a system for automatically converting Russian to morphemes, a kind of base syllable, was created and tested. A suite of tools for front end processing, automatic forced alignment, and a complete automatic recognition system are described.

15. SUBJECT TERMS

Broad phonetic classes, Forced Alignment, Front-end Feature Extraction, Speech Recognition

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			**STANLEY J. WENNDT**
U	U	U	UU	137	19b. TELEPHONE NUMBER (*Include area code*) **N/A**

Table of Contents

LIST OF FIGURES

LIST OF TABLES

1. SUMMARY

This report gives a detailed summary of research work completed under Air Force Research Laboratory (AFRL) Award No. FA87501210093, "Detailed Phonetic Labeling of Multi-Language Database for Spoken Language Processing Applications," over the time period (Jan 19, 2012 – Jan 18, 2013).

2. PROJECT INTRODUCTION

The main goals of this project were to investigate and test methods and tools to enable more accurate phonetic labeling of speech databases in various languages. Such methods and tools are valuable for developing automatic speech recognition (ASR) systems in these languages. One of the biggest hurdles to further improvements in ASR accuracy and robustness is the need for really large speech databases in multiple languages. One of the big assets to ASR research is the availability of large open source databases along with text annotations at the sentence level. However, for ASR applications, this text information should be "force" aligned with the acoustic signal, in terms of an accurate phonetic level transcription. In this work, we focus on the front end signal processing to improve phonetic level recognition, with experimental work in three languages- - English, Mandarin, and Russian. The report contains six major sections, each with its own introduction and conclusion. This report also contains copies of four conference papers and 1 conference abstract, for work supported by this project. In the remainder of this overall introduction, we give a brief executive summary of the major sections of this report.

Section 3, "Generalized spectral-temporal features for representing speech information," reports a detailed mathematical and experimental investigation of a method for computing speech features as weighted sums of spectral values about each instant in time. This method is viewed as a general framework which includes the commonly used MFCCs (or other filterbank implementations in place of the Mel filterbank used to compute MFCCs), delta, and delta-delta, features. Some other frontend methods, such as perceptual linear prediction (PLP) and Gabor filtering, although are not entirely covered mathematically by this framework, can still be studied using the unified basis vector point of view, which reveals the essence of features as linear transformations of the spectrum. Although these weighted sums are most generally characterized as two dimensional basis vectors over time and frequency, the most effective implementation found is based on two one-dimensional sets of basis vectors. The first step, summing over frequency, is referred to as a DCT (Discrete Cosine Transform), with resulting features called DCTCs (last C being Coefficient). The DCTCs are then summed over time to create DCSCs (Discrete Cosine Series Coefficients). Using DCTC/DCSC features and an HMM recognizer, the highest accuracy phonetic recognizer obtained with English (74%) is an improvement over the best accuracy obtained with the more typical 39 MFCC feature set (71.4%).

Section 4, “Automatic word to morpheme decomposer for Russian,” reports progress on an approach to ASR for Russian. Unlike English, and many other western European languages, Russian has a very unconstrained grammar system (many allowable word orderings, with essentially the same meaning) and an extremely large word vocabulary. Words can however be decomposed into base units called morphemes, consisting primarily of prefixes, roots, and suffixes. The base part of the meaning is the root. There are orders of magnitude fewer roots than words in Russian, and one approach to Russian ASR would be to first recognize morphemes (especially roots) and then reconstruct words from morpheme strings and grammar rules. However, to develop a Russian ASR system, an automatic word to morpheme convertor is needed for training purposes. Although such convertors exist they are all proprietary, not well described, and not available for general use. In this section of the report, a word to morpheme convertor is described and developed using open source Matlab code. Limited experimental tests have shown the convertor to be extremely accurate.

Section 5, “Non-uniform frame spacing for speech feature calculations,” is a report on our efforts to improve HMM based ASR accuracy by computing features with variable frame spacing. It is quite clear from basic principles of speech production that some temporal regions in a time-frequency representation of speech are changing quite rapidly (for example stop consonants) whereas other regions (especially vowels) are changing much more slowly. Although one approach to improving accuracy would seem to be to sample features at a fast enough rate to accommodate the rapidly changes regions of the short time spectrum, in practice, at least with HMM recognizers, this “oversampling” actually degrades accuracy. In this section of the report a few methods for non-uniform sampling of features and corresponding experimental results are summarized. Unfortunately none of these methods proved useful in terms of improving ASR accuracy. However, we still speculate that better measures of spectral change coupled with modifications to the recognizer itself, have promise for this general approach.

Section 6, “A toolbox for a complete automatic speech recognition system,” describes a suite of tools for developing an automatic speech recognition system. The kernel of the toolbox is the Hidden Markov Model ToolKit Version 3.4 (HTK 3.4). The framework is developed in Matlab running under the MS windows operating system. Key steps include forced alignment (if needed), front end feature calculations, acoustic model training, including monophone and triphone models, language modeling (two options available, one for building a simple bigram, another for more complex n-gram tasks), and decoding (which provides two decoders, one for a small vocabulary task, one for a large vocabulary task).

Section 7, “Forced alignment tool package user manual,” is a suite of tools, also based on Matlab and the HTK, which describes in detail the use of the forced alignment part of the ASR toolbox, from section 6. This toolbox creates accurate phonetic labelling of pronunciations from the raw wave files and word level transcriptions. The output of this toolbox is used in subsequent tools.

Section 8, “A tool for speech feature extraction--tfrontm guide,” describes the setup and use of the front end analysis package which can be used with the ASR toolbox, from section 6. This tool is completely written in Matlab. The center pieces is DCTC/DCSC analysis, as described in section 3, but there are also options for MFCCs, deltas, pitch tracking, and other front end analysis types.

3. GENERALIZED SPECTRAL-TEMPORAL FEATURES FOR REPRESENTING SPEECH INFORMATION

3.1. Introduction and background

Over many years, for both automatic speech recognition (ASR) and general speech science applications, there is an ongoing search for “good” acoustically derived features. The meaning of “good” depends on the particular intended usage, but generally includes the following elements:

1. Relevance: Generally, speech features should closely reflect the main characteristics of speech activities for speech production and/or speech perception. For the case of features intended for use in speech science or speech therapy applications, this property is very important. For ASR applications, this property is only indirectly important. Presumably, but not necessarily, features which mimic human processing of speech will also be more effective for ASR.

2. Compactness: Due to issues such as the “curse of dimensionality" [1] in obtaining robust estimates of parameter distributions in various tasks, such as speech recognition, speaker identification, etc., the information in speech should be encoded with a relatively small number of features.

3. Completeness: The features should represent all speech information of interest.

4. Robustness: “Good" speech features should have very similar values for similar sounding speech, even in the presence of noise. Humans clearly have the ability to recognize speech sounds under a broad range of conditions, including distortion and noise. For ASR, a goal is that recognition be accurate under a similar broad range of conditions.

In this section of this report, a speech frontend is presented which extracts speech features guided by the above general principles, and for which tradeoffs can easily be explored to tune features for best “performance.” The primary tradeoffs explored are between time and frequency resolution, a fundamental issue in short time spectral analysis.

As mentioned above, speech production and perception aspects of speech science form the foundation for signal processing to extract speech features. In terms of speech production, by far the most widely used acoustic features for characterizing vocal tract shape are formants. According to the classic Peterson and Barney's vowel study [2],

formants, corresponding to resonant frequencies (poles) in the vocal tract, are most effective for distinguishing vowels. However, for many more complex phones, such as fricatives, nasals, and stop consonants, poles of the vocal tract transfer function are not sufficient, and zeros must be considered. For ASR applications, guided by the underlying importance of the formants, and the signal processing idea that nearly any transfer function can be approximated by a high order all pole model, the vocal tract is greatly simplified to an all-pole system, such as in the Perceptual Linear Predictive (PLP) frontend [3]. More typically, however, for ASR, a pole or pole-zero approximation to the vocal tract model is replaced by cepstral features [4], which primarily extract vocal tract information from the log magnitude of the spectral envelope, and remove voice source information, and thus, implicitly encode the formants.

Speech perception research, both from the physiology and psychoacoustic fields, establishes the fundamental theory for many widely used speech frontends. Typically, frequency and time resolution of speech perception are the dominant two considerations that a frontend is designed upon. As pointed out in Zwicker's work [5], frequency resolution stems from the cochlea's frequency selectivity properties: a sound wave, when travelling along the basilar membrane, causes maximum displacement of the membrane oscillation at different positions and with different resolutions for different frequency components. This physiological property, leads to the development of various perceptual frequency scales, and these perceptual scales, in turn, lead to the use of auditory filter banks to mimic the frequency selectivity of human ears.

As one example, Mel Frequency Cepstral Coefficients (MFCCs), proposed by Bridle and Brown [6], are widely used as speech features. Figure 1(a) is a block diagram of a typical MFCC frontend. After being pre-emphasized and taking the Short Time Fourier Transform (STFT), a speech signal is filtered through a set of triangular filters which evenly partitions the Mel scale into 26-30 equal-width bins. On the Mel scale, according to Stevens et al. [7], the "perceived frequency" (pitch) becomes linear. For example, psychologically, a pitch of 1000 Mels is 2 times higher than a pitch of 500 Mels. The bandwidths of these filters reflect perceptual frequency resolution. The nonlinear logarithmic-like frequency “warping” function that maps the Hertz scale to the Mel scale results in a higher bandwidth in the Hertz domain as frequency increases. Thus, the frequency resolution decreases at high frequencies, which means that a wider range of frequency components are perceptually nearly identical. The output power of each filter channel is then nonlinearly amplitude scaled to convert the physical loudness to the psychologically perceived loudness, which is linearly proportional to the neuron firing rate of the auditory nerves [8]. Then, these amplitude-scaled speech powers on the Mel scale are converted into a set of cepstral features by taking the Discrete Cosine Transform (DCT), which also eliminates the heavy correlations among the filter bank output powers due to the overlapping of the channels.

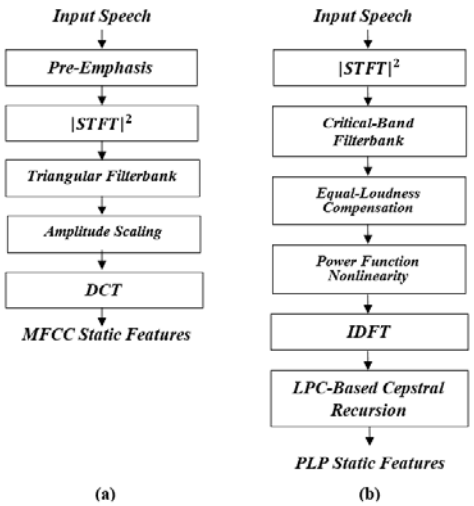

Figure 1: Comparison of the MFCC and PLP structure. These block diagrams produce static features. The first and second order differentials of the static features are usually appended as dynamic features.

Another fundamental perceptual scale which is used to create auditory filter banks is the Bark scale. This scale is based on the concept of Critical Bands in Fletcher's canonical paper [9], in which he first pointed out that frequency components in the same critical band are perceptually nearly indistinguishable. One popular use of the Bark scale is the PLP frontend developed by Hermansky [3]. Figure 1(b) depicts the block diagram of this frontend. Suggested by Hermansky, the complex frequency components in the speech are filtered through typically 16 trapezoids uniformly spaced on the Bark scale, with the bandwidth of each channel approximately 4 Barks. Though the Bark frequency warping and the Mel frequency warping are derived using very different psychoacoustic methods, they both have high frequency resolution at low frequencies and low frequency resolution at high frequencies. Unlike the MFCC frontend, in the PLP framework, the pre-emphasis step is performed by an equal-loudness compensation for each filter channel output as an approximation to the nonequal sensitivity of human hearing at different frequencies [10]. Another salient difference from MFCCs is the Linear Predictive (LP) processing in the Bark domain, which explicitly models the formant positions of the vocal tract by a set of LP coefficients computed by Durbin's recursive method [11], and finally, these LP coefficients are converted into a set of cepstrum coefficients by another recursion [12]. As a variant to the Bark scale, Moore and Glasberg proposed the Equivalent Rectangular Bandwidth (ERB) scale [13], and they showed that the auditory frequency resolution can be more precisely described by the ERB scale than the Bark scale based on the famous "notched-noise data" experiments. Based on this scale, gammatone filter banks were first proposed by Patterson et al [13], and Slaney in [14], proposed an efficient implementation of the gammatone filter bank by viewing each filter as a cascade of four 2nd order filter stages.

These perceptual scales for frequency only capture the static frame-based frequency information of the speech spectrum. They do not characterize spectral trajectories over time. However, in principle, a sufficiently large set of frequency-based features, plus voice source information, should contain nearly all speech information, since high quality speech can be synthesized from these static features, with no need for temporal

information. Thus, for ASR, at least in principle, a powerful recognizer should also be able to extract the dynamic trajectory and make decisions from static features alone. Although the fundamental strength of HMMs (that is Hidden Markov Models, the dominant and effective method for modeling acoustic phonetic information in ASR systems) is the ability to recognize patterns in temporal sequences of variable length, apparently the HMM framework is not able to adequately capture the patterns contained in sequences of static speech features alone. The dynamic features appear to make these static patterns more easily discernible. For the case of HMM ASR systems, dynamic features can be viewed as a compensation for a deficiency in HMMs.

In [15,16,17,18], a simple but very effective type of dynamic features is obtained by computing the time "derivatives" of the static features. Empirically, it has been determined that ASR performance improves considerably (reduction in error rate on the order of 20%) if the so called delta and acceleration (the second order differential) terms are appended to the static features. Mathematically, the delta terms are computed as:

$$\Delta_t = \frac{\sum_{\theta=1}^{\Theta} \theta(c_{t+\theta} - c_{t-\theta})}{2\sum_{\theta=1}^{\Theta} \theta^2} \quad (1)$$

where Δ_t is the differential at time t computed in the context from the static coefficients $c_{t-\theta}$ to $c_{t+\theta}$, with $2\Theta + 1$ being the window length. It can be seen that this method simply estimates the continuous time derivative at each time instant using its discrete time approximation. It does not account for the non-uniform time resolution of the human auditory system.

Spectral-temporal modulation features are much more effective than the delta method in solving the problem of non-uniform time-frequency resolution as well as efficient sampling of short time representation. In 1994, Drullman et al. [19] found that the most important spectral trajectory information over time for speech perception was in the range of 1-16Hz modulation frequencies. Highlighted by this finding, conceptually, modulation features are typically computed by first dividing the spectrum into frequency bands, and then representing the trajectory envelope of each band by temporal features. In order to exploit the information at the modulation frequencies, relatively long time blocks are analyzed. The modulation features of each band are usually either fully appended to the static features, or selected by algorithms. To encode the temporal trajectory of spectral bands by modulation features, various methods have been proposed. In [20], Athineos et al. use the dual of conventional linear prediction in the time domain for each sub-band to model the poles of the temporal envelope. In [21], Valente and Hermansky developed a hierarchical and parallel scheme combining independent classifier outputs and modulation frequency channels.

More recently, Gabor-filter-based approaches to extracting modulation features with direction oriented time-frequency resolution have been proposed. Gabor filters are defined using the product of a two-dimensional Gaussian envelope and a complex exponential function with a localized region in the time-frequency plane. The strength of the Gabor filter bank is that it captures Localized Spectro-Temporal Features (LSTFs) as suggested in [22,23]. However, the large number of parameters, which allow Gabor

filters to be tuned toward different directions of the spectral-temporal modulation, present the problem of how to select the most effective and compact feature set for ASR. One feature selection method proposed in [24] uses a Feature-Finding Neural Network (FFNN). The importance of each feature is evaluated by the increase of RMS classification error after its removal from the feature set. Though the linear classifier has a relatively fast converging rate, the training process is still much slower than that for other conventional ASR feature extraction methods.

Based on this prior extensive ground work, this section presents a generalized spectral-temporal feature extraction frontend for representing speech information. This feature set presents a detailed look at one general flavor of time-frequency features, focusing on the primary properties of human hearing: frequency and time resolution, but which encompasses quite a range of time-frequency representation options. It's not just one specific type of frontend, but should be viewed as a unified framework that realizations of the general time-frequency concepts can be easily implemented and tuned. Based on a set of frequency warping and frequency-dependent time warping functions, it's flexible enough to easily evaluate the relative importance of the spectral and temporal features, and to explore the trade-off between frequency selectivity and time resolution. In addition, a wide range of conventional filter bank-based static features as well as the time derivative dynamic terms can be easily incorporated into this generalized framework by modifying the basis vectors. Thus, it provides a common yardstick to study, compare and develop different time-frequency representations.

3.2. Method

In general terms, the spectral-temporal features that are the primary focus of this section, are viewed as weighted sums of short time spectral magnitudes, computed at each (sample) instant of time, from all short-time spectral frequency components in a temporal region centered at each sample instant in time. Figure 2(a) presents a high level diagram of the proposed frontend. After an utterance has been pre-emphasized and segmented into frames, a time-frequency representation (TFR) of the speech, denoted by $X(t,f)$ is obtained by computing the magnitude-squared Short-Time Fourier Transform (STFT). In this work, the notation t and f denote the physical time (in seconds) and frequency (in Hertz). The STFT has uniform frequency and time resolution over the entire time-frequency plane, determined by the analysis window shape and width [25]. This representation does not take into account the non-uniform perceptual scale of the peripheral auditory system. Thus, we first define t' and f' as perceptual time and frequency scales, whose desirable properties will be described in detail. Then, a set of features $Feat(i,j)$ for the time block centered at time instant t, can be expressed as:

$$Feat(i,j) = \int_{t'=-1/2}^{1/2} \int_{f'=0}^{1} a(X'(t',f')) \cdot BV_{i,j}(t',f') df' dt' \qquad (2)$$

In Eq.2, the feature extraction is performed entirely using perceptual scales, where $X'(t',f')$ is the power spectrum of a time-frequency block on this domain, in which the frequency f' has been normalized to the range of {0,1} by subtracting an offset and

dividing by a scaling factor. Similarly, the perceptual time t' has been normalized to the range of $\{-1/2, 1/2\}$, with $t' = 0$ corresponding to the center of the time block, which typically spans approximately 200ms (unscaled physical time). The function $a(\cdot)$ nonlinearly maps the power spectrum to a psycho-loudness scale, typically using a logarithmic scaling, or a power-law nonlinearity [26]. Finally, the amplitude-scaled power spectrum is weighted by a set of two-dimensional basis vectors $BV_{i,j}$ also defined for the perceptual domain (t', f'). The number of features extracted from a time-frequency block depends on the number of basis vectors used.

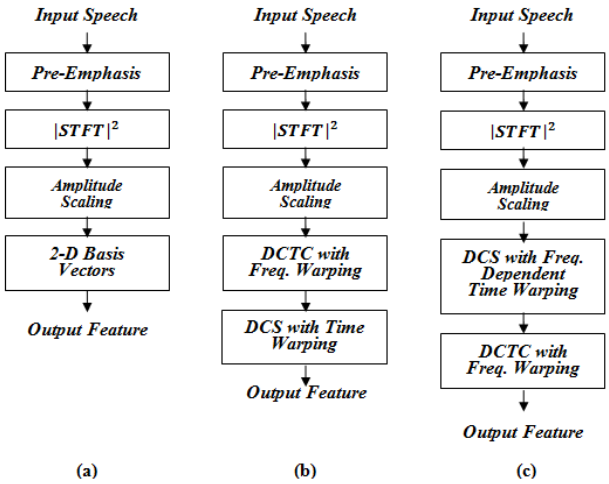

Figure 2: Block diagrams of the proposed frontend. (a). The amplitude-scaled power spectrum is weighted by a set of 2-D basis vectors. (b). The DCTC-DCS implementation. In this method, DCTCs are computed first, followed by DCS. The time warping in the DCS basis vectors is uniform for all frequencies. (c). The DCS-DCTC implementation. A set of DCS coefficients is obtained, followed by DCTC. This implementation enables frequency-dependency in the DCS basis vectors.

In this section, a set of two-dimensional cosine basis vectors for $BV_{i,j}$ are used to compactly encode the spectral envelope as well as the spectral trajectory, and also to de-correlate the features. Specifically, the 2-D cosine basis vectors operating in the perceptual space are defined as:

$$BV_{i,j}(t', f') = cos(\pi i f') \cdot cos(\pi j t') \tag{3}$$
$$0 \le i \le \text{number of frequency bins}, \quad 0 \le j \le \text{block length in frames}$$

From Eq. (2) and (3), it can be seen that the warping from f to f' and t to t', together with their derivatives df' and dt' characterize the desired frequency and time resolution of human hearing. The following few paragraphs mathematically show how the nonlinear mappings are incorporated into the feature calculations.

Considering frequency variables first, a frequency warping, which specifies the relation between the perceptual frequency f' and the physical frequency f is defined:

$$f' = g(f), \; 0 \le f \le 1 \tag{4}$$

where the physical frequency range has also been normalized to $\{0,1\}$[1]. Thus, the df' term in Eq. (2) is equivalent to:

$$df' = \frac{dg}{df} df \tag{5}$$

As per the discussion in a previous section, one reasonable choice for the form of the frequency warping $g(f)$, is a Mel-shape warping defined as:

$$g(f) = C \cdot log_{10}(1 + \frac{f}{k}) \tag{6}$$

where k is an adjustable warping factor between 0 and 1 that controls the degree of the warping, and the constant C is chosen to insure that $f = 1$ is mapped to $f' = 1$. It's easy to see that this Mel-shape warping becomes the normalized version of the most widely used "standard" Mel warping proposed by O'Shaughnessy [27] if k=0.0875 and C=0.9137 for the frequency range of 0 to 8000Hz. Another option, according to Smith and Abel's work [28], is to use a bilinear warping to mimic the Bark scale, as per:

$$g(f) = f + \frac{1}{\pi} tan^{-1}\left\{\frac{\alpha \cdot sin(2\pi f)}{1 - \alpha \cdot cos(2\pi f)}\right\} \tag{7}$$

with α being the warping factor ranging from 0 to 1. In Figure 3, a family of five bilinear warpings with different α values are plotted, starting from $\alpha = 0$, and gradually increasing. For comparison, the normalized Mel warping using O'Shaughnessy's equation in [27], and the normalized Bark warping according to Wang et al. are also depicted [29]. The Mel and the Bark warping can be closely approximated by the bilinear warping using appropriate warping factors.

[1] In the derivation of the frequency warping *g*(*f*) and its derivative, for convenience, the normalized frequency range {0,1} of *f* corresponds to the full un-normalized range {0, *Fs*/2} where *Fs*/2 is the Nyquist frequency. The normalized perceptual frequency *f*' in {0,1} corresponds to the range of 0 to the perceptual frequency of *Fs*/2. In practice, if the desired frequency range is not {0, *Fs*/2}, one can simply extract the corresponding segment of the full warping curve defined in Eq.(6) or (7), as well as their derivatives, and re-normalize this segment of *g*(*f*) to the range of {0,1}.

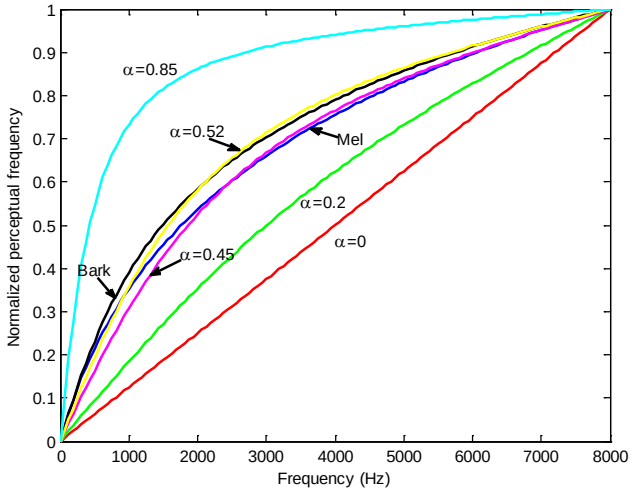

Figure 3: Normalized bilinear warping with different warping factors. Mel and Bark warping shown for comparison.

From Eq. (5), in contrast to using an auditory filterbank to determine frequency selectivity, such as in the MFCC, PLP or gammatone frontends [3,6,13], the derivative of the frequency warping determines the frequency resolution for each frequency. In the filterbank methods, the range of perceptually indistinguishable frequency components that fall into a channel is quantified by the filter bandwidth, which in turn is determined by the warping function. So, we can view a filterbank as a quantizer which partitions the perceptual frequency scale into a finite number of equal but coarse intervals. In the proposed approach, this quantization is effectively continuous (in practice limited only by the frame length of each analysis window and the spacing of FFT samples used to compute the original spectrum). The frequency selectivity is reflected by the derivative term *dg*(*f*)/*df* in a straightforward way.

The frequency selectivity term can be interpreted from a physiological perspective. The term *dg/df* can be thought of as how far the maximal displacement position advances along the basilar membrane caused by a 1 Hz change at frequency *f*. On the physical f scale, the same Δf corresponds to small displacement at high frequencies versus large displacement at low frequencies. Thus, the frequency resolution is non-unifrom, as defined by *dg/df*. However, on the perceptual f' scale, this non-uniformity vanishes since frequency components in the same $\Delta f'$ covers approximately the same range along the basilar membrane for both low and high frequencies.

Next, the relation between the perceptual time t' and linear time t is specified through a time warping with the range of t also normalized, but to the range $\{-1/2, 1/2\}$:

$$t' = h(t, f) \qquad -\frac{1}{2} \le t \le \frac{1}{2}, \quad 0 \le f \le 1 \qquad (8)$$

The perceptual time t' can be viewed as a psychological time scale that defines a "pseudo" time instant at which an acoustic event (such as the evolution of the spectral envelope over time) that happens at physical time t is perceived by the auditory neurons.

The insight of this perceptual time is more clear and relevant in terms of its derivative with respect to t:

$$dt' = \frac{dh(t,f)}{dt} \tag{9}$$

This time resolution term indicates how far apart two events, separated by unit time (1 sec) on the linear scale, are when they are perceived. A large value means that two acoustic events are clearly psychologically distinguishable whereas a small value shows that the time boundary between events is blurred, and thus cannot be well resolved. When characterizing the temporal trajectory of a frequency component f over the period of a time block, it's reasonable to use non-uniform time resolution by imposing higher resolution near the center of the block, which is the current "observation" frame, than at the far ends. This can also be explained by the relative importance of the spectral information: to identify the content of the current frame with the help of its left and right context, it is reasonable to assume high relative importance for contexts close to the current frame than for those far it. Hence, important temporal changes of the spectrum envelope need to be clearly resolved, whereas less helpful parts are suppressed. Therefore, in this work, the choice of the shape for *dh/dt* is approximately Gaussian over a time block for each frequency. In the experimental work, a Kaiser window for *dh/dt* is used, which has one parameter, defined as the time warping factor, that conveniently controls the degree of warping.

Note that in Eq. (8), the time warping is shown with dependence on frequency f, and so is the case for the time resolution in Eq. (9). This allows an exploration of the trade-off between frequency and time resolution, including making these tradeoffs frequency dependent, which is a basic property of the peripheral auditory filters. Based on the psychoacoustic masking experiments in [30], the very narrow bandwidth at low frequencies produces high frequency resolution, but also prolongs the "ring" time at the onset and offset transients for short signals, and thus degrades the time resolution of the excitation patterns. This trade-off is also justified in [31] by neurophysiological experiments and in [32], by the gap-in-noise detection experiments, which provides evidence that human subjects are able to detect short gaps with higher time resolution between narrow band noise segments with increasing center frequencies. Despite these properties of human hearing, it's not yet clear whether this trade-off has a significant impact on ASR. Our work provides one way to account for and examine this effect by incorporating frequency-dependency in the time warping. Specifically, the shape for the term *dh/dt* can be made more “peaky” at high frequencies than at low frequencies. Figure 4 plots the desirable time resolution using a Kaiser window with different warping factors for different frequencies. The time resolution is non-uniform both over time within a block and over frequency.

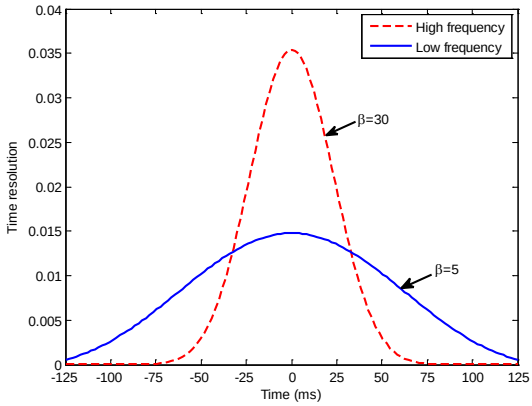

Figure 4: Desirable time resolution dh/dt for low and high frequencies using a Kaiser window. The time resolution is non-uniform over both time and frequency.

Now that the principles and reasonable forms of the frequency and time warping have been established, also note that the magnitude of the power spectrum on the perceptual scale is the same as it is on the physical time-frequency domain, and with t and t', f and f' being normalized to the same range respectively, Eq. (2) can be rewritten in terms of t and f by substituting in Eq. (3)(4)(5)(8)(9):

$$Feat(i,j) = \int_{t=-1/2}^{1/2} \int_{f=0}^{1} a\big(X(t,f)\big) \cdot cos\big(\pi i g(f)\big) \frac{dg(f)}{df} \cdot cos\big(\pi j h(t,f)\big) \frac{dh(t,f)}{dt} df dt \tag{10}$$

Eq. (10) can be more conveniently interpreted by defining modified basis vectors over frequency f (compared to the original form over f') as:

$$\varphi_i(f) = cos\big(\pi i g(f)\big) \frac{dg(f)}{df} \tag{11}$$
$$0 \le i \le \text{number of frequency bins}$$

and modified frequency-dependent basis vector over time t as:

$$\psi_j(t,f) = cos\big(\pi j h(t,f)\big) \frac{dh(t,f)}{dt} \tag{12}$$
$$0 \le j \le \text{block length in frames}$$

Using the basis vectors in Eq. (11) (12), Eq. (10) can be rewritten as:

$$Feat(i,j) = \int_{t=-1/2}^{1/2} \int_{f=0}^{1} a(X(t,f)) \cdot \phi_{i,j}(t,f) df dt \tag{13}$$

where the modified two-dimensional basis vectors $\phi_{i,j}(t,f)$ is the product of the basis vectors in Eq. (11) and (12).

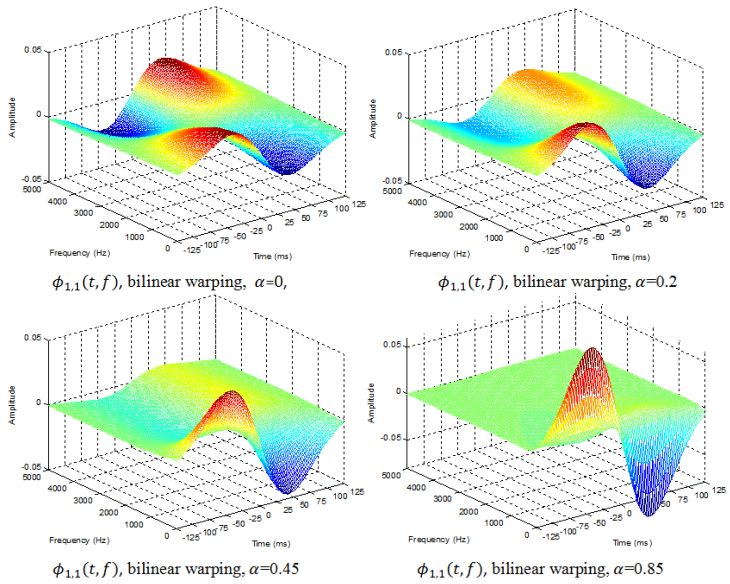

Figure 5: Two-dimensional basis vector ϕ_(1,1) (t,f), with uniform time warping over all frequencies using a Kaiser window with β=5, but a bilinear frequency warping with varying degree.

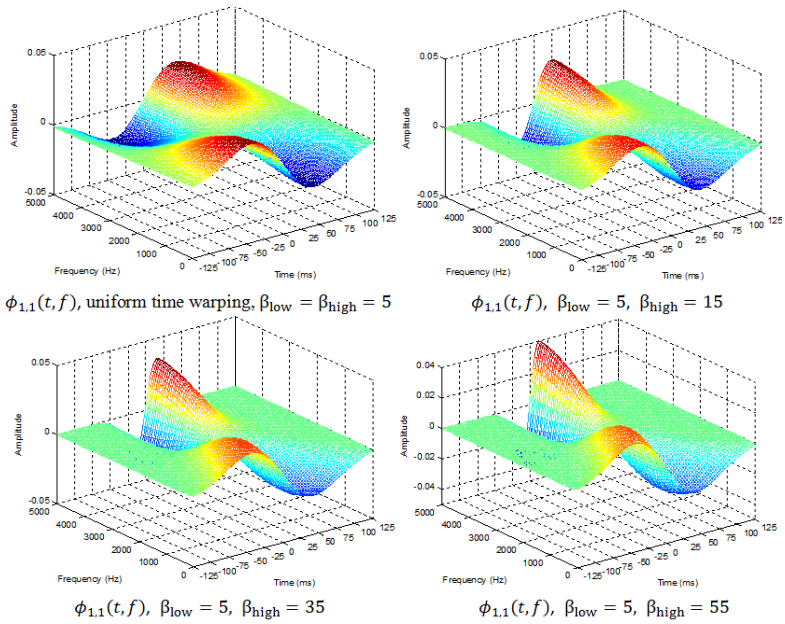

Figure 6: Two-dimensional basis vector ϕ_(1,1) (t,f) with linear frequency scale but frequency-dependent time warping using a Kaiser window. β_low and β_high are the low and high frequency warping factors.

In Figure 5, the two-dimensional basis vector $\phi_{1,1}(t,f)$ is plotted with uniform time warping over the entire frequency range, using a Kaiser window with $\beta = 5$, but with various values of α, i.e. the degree of the bilinear frequency warping. It can be seen that starting from the linear frequency scale which has uniform resolution, the basis vector becomes more sharply peaked at low frequencies as we impose higher frequency resolution at low frequencies through a larger warping factor, i.e. higher *dg/df* values. In Figure 6, the same basis vector is plotted using increasing time warping as frequency

increases. The Kaiser window β value is linearly interpolated between β_{low} and β_{high}. The higher time resolution for high frequencies makes the basis vector more concentrated near the center of the block.

Another option for the two-dimensional basis vectors, rather than the cosine expansion, is to use a Gabor filterbank. As described in the work of [22,23,33], Gabor filtering is performed by a two-dimensional correlation (which is slightly different than the integration operation in this work) between the Gabor filterbank and the perceptual time-frequency plane (t', f'). Each filter is defined as:

$$G(t', f') = n(t', f') \cdot e(t', f') \tag{14}$$

where $n(t', f')$ is a Gaussian envelope centered at (t_0', f_0'):

$$n(t', f') = \frac{1}{2\pi\sigma_f\sigma_t} exp\left[\frac{-(f' - f_0')^2}{2\sigma_f^2} + \frac{-(t' - t_0')^2}{2\sigma_t^2}\right] \tag{15}$$

and $e(t', f')$ is the complex Euler function:

$$e(t', f') = exp\left[j\omega_f(f' - f_0') + j\omega_t(t' - t_0')\right] \tag{16}$$

The width of the Gaussian envelope is defined by σ_f and σ_t, and the modulation frequencies ω_f and ω_t modulates the filter in particular directions. In Figure 7, we plot the real part of a group of Gabor filters on the physical time-frequency domain centered around 1000Hz and zero time instant, with various directions tuned by the modulation frequencies. Indeed, directionality is the most salient difference between Gabor filters and the cosine expansion used in this section. Gabor filters can be adjusted towards any directions whereas the cosine transform only represents modulation of the spectrum along the vertical and horizontal axis (compare Fig.7 with Fig.5,6). The deeper reason is that the Gabor approach and the method presented in this section model very different auditory properties. The directionality of the Gabor frontend stems from the response of the neurons to combinations of spectral-temporal modulation frequencies in the spectral-temporal receptive field [34], whereas the proposed framework in this section aims to model the time-frequency tradeoff of the peripheral auditory system.

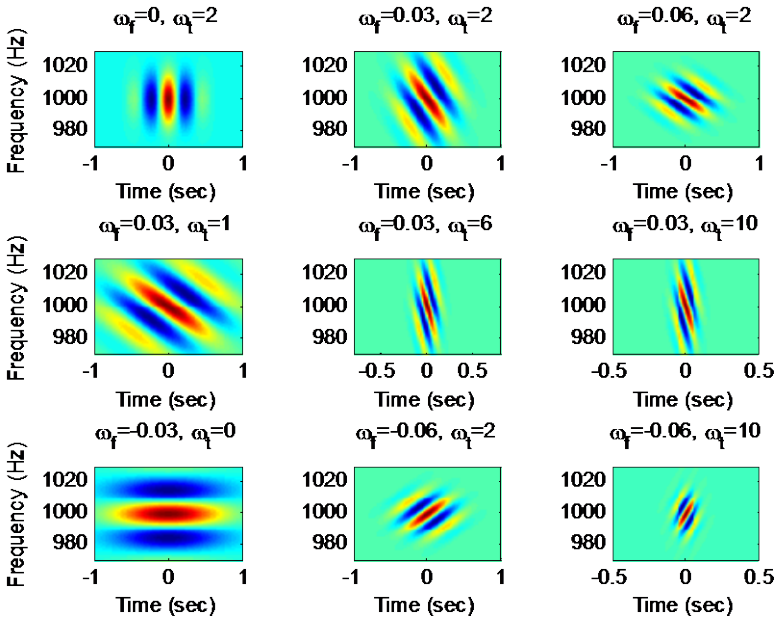

Figure 7: Real part of Gabor filters in the physical time-frequency plane. Directions are determined by ω_f and ω_t. The unit of ω_t/2π is Hz, and the unit of ω_f/2π is cycles/Hz.

As mentioned before, a big drawback of the Gabor frontend is the extremely large number of features typically computed with this approach. Feature selection is usually conducted by a Feature Finding Neural Network [24]. In addition to the obvious issue of computational time, another "pitfall" is that feature selection may result in an optimal set only for the database used during this selection. The proposed frontend in this work does not have these issues, since the 2-D cosine basis vectors compactly and efficiently encode both the spectral envelope as well as the spectral trajectory with appropriate resolution, thus yielding a relatively small number of features for each block. However, it is feasible to modify the proposed frontend to account for the directionality of spectral-temporal patterns in a similar but not totally identical way as the Gabor filterbank (the modified features using cosine basis vectors are not as "localized" as the Gabor features). In our prior work [35], this is achieved by the idea of rotating the 2-D cosine basis vectors by various angles.

3.3. Implementation

The 2-D integral in Eq. (10) can be implemented in various ways, depending on the order of the two 1-D integrations. All the continuous integrations are carried out by vector inner product between basis vectors and the sampled time-frequency plane. First, if the dependence on f is omitted in the time warping, i.e. uniform time warping for all frequency components, integrating by any order (first over f, and then over t, or the reverse) are equivalent. Conventionally, frequency integration is performed first, which generates a set of static features called Discrete Cosine Transform Coefficients (DCTCs):

$$DCTC(i) = \int_{f=0}^{1} a(X(t,f)) \cdot \varphi_i(f) df \tag{17}$$

where $\varphi_i(f)$ is the ith static basis vector, as defined in Eq. (11). Then, the trajectory of these DCTCs is encoded by the integration over time, yielding a set of Discrete Cosine Series Coefficients (DCSCs), which are also referred to as dynamic features:

$$DCSC(i,j) = \int_{t=-1/2}^{1/2} DCTC(i) \cdot \psi_j(t)dt \tag{18}$$

where $\psi_j(t)$ is the jth dynamic basis vector, as defined in Eq. (12) with the dependence on f being removed. These DCSCs features are then input to the recognizer. The diagram of this implementation was plotted in Figure 2(b). Figure 8 depicts the first three DCTC and DCSC basis vectors, using a Mel-shape and a Kaiser window of $\beta = 5$ for frequency and time warping respectively. The zeroth terms represent the envelope of the basis vectors, which also define the spectral/temporal resolution.

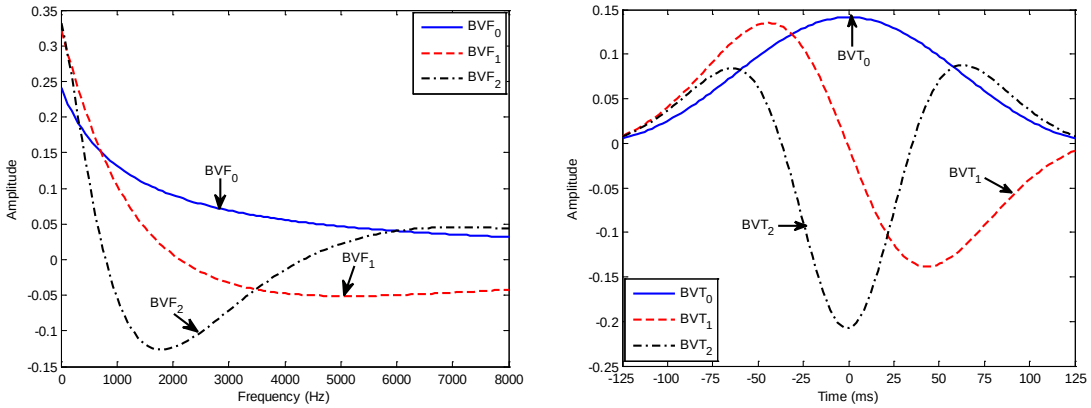

Figure 8: The first 3 DCTC (left-a) and DCSC (right-b) basis vectors. A Mel-shape and a Kaiser window are used for frequency and time warping respectively.

Unlike some other spectral-temporal modulation frontends, such as RASTA [36], TRAPS [37], as well as the Gabor method mentioned in the previous section, in which modulation frequencies are explicitly defined, the DCTC and DCSC basis vectors in the proposed frontend do not use this concept. However, the DCSC basis vectors achieve similar effects as non-causal FIR low pass temporal filters, by encoding the temporal evolution of integrated spectral dynamics. Similarly, the DCTCs can also be viewed as outputs of spectral low pass filtering. Based on this idea, parameters in the DCTC/DCSC implementation can be flexibly varied to examine the trade-off of the overall spectral-temporal resolution. It should be pointed out that the meaning of the "overall" spectral and temporal resolution being discussed at this point is somewhat different than the auditory time-frequency resolution built into the warping of the basis vectors, as presented in previous sections. Here, the overall spectral-temporal resolution, based on the filtering point of view, can be interpreted as how much detail of the static spectrum and dynamic trajectory are preserved after the low pass filtering, whereas the time-frequency resolution represented by the derivatives of the warping (which, as discussed in previous sections, also poses a trade-off effect) is an intrinsic property of human hearing. As mentioned, the proposed DCTC/DCS frontend can be tuned to emphasize either side of the overall spectral or temporal resolution. For increased emphasis on the spectral information, a long frame length and a relatively large number of DCTCs can be

employed, with a relatively small number of DCSCs computed from a long block length, whereas for increased emphasis on the overall time resolution, a short frame length and frame spacing can be used with a relatively large number of DCSCs computed from a relatively short block length.

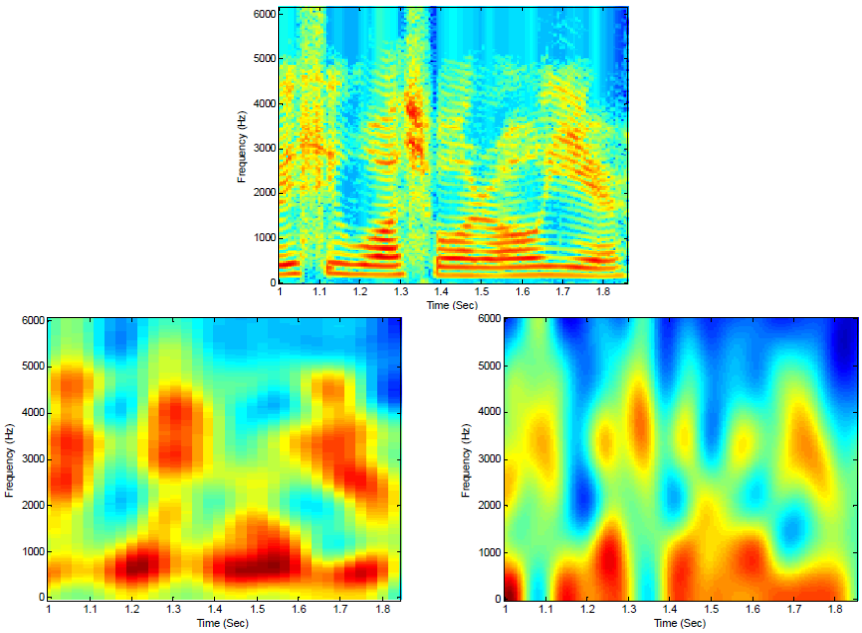

Figure 9: Spectrogram of a speech segment (upper panel) and two rebuilt spectrograms. The bottom left one has high spectral resolution and low temporal resolution while the bottom right one has low spectral resolution but high temporal resolution.

Figure 9 graphically illustrates this spectral-temporal trade-off. The top panel depicts the spectrogram of a speech segment; two rebuilt spectrograms, from DCTC/DCSC terms, are presented in the bottom panels. The left one has high spectral resolution but low temporal resolution. It is rebuilt using 16 DCTCs, which are computed with a 25ms frame length, a 10ms frame space and 4 DCSCs using a block length of 50 frames (500ms). The one in the right bottom panel has low spectral resolution, but high temporal resolution. It is computed from 8 DCTCs with a 5ms frame length, spaced by 2ms, and 6 DCSCs with a block length of 100 frames (200ms). The low frequency components in both rebuilt spectrograms are represented with relatively higher auditory frequency resolution due to the Mel-shape warping. Comparing the two rebuilt spectrograms, the left panel preserves more spectral details in each frame whereas in the right panel, most spectral details are eliminated. Also, the spectral trajectory in the left panel is smeared, while in the right panel, the trajectory is rebuilt with much more detail.

In the case where the frequency-dependence remains in the time warping, as originally proposed in the previous section, the 2-D integration in Eq. (10) can be implemented by integrating over the time axis first, followed by another integration over frequency. Figure 2(c) depicts the diagram of this configuration. In this case, Eq. (10) can be rearranged as:

$$Feat(i,j) = \int_{f=0}^{1} cos(\pi i g(f)) \frac{dg(f)}{df} df \int_{t=-1/2}^{1/2} a(X(t,f)) \cdot cos(\pi j h(t,f)) \frac{dh(t,f)}{dt} dt \quad (19)$$

The inner integral defines a set of frequency-dependent DCSCs:

$$DCSC(j,f) = \int_{t=-1/2}^{1/2} a(X(t,f)) \cdot \psi_j(t,f) dt \quad (20)$$

where $\psi_j(t,f)$ is the *j*th DCS basis vector for frequency *f*, as defined in Eq. (12). Then, the outer integral over frequency computes the DCTCs, which yields the final features:

$$Feat(i,j) = DCTC(i,j) = \int_{f=0}^{1} DCSC(j,f) \cdot \varphi_i(f) df \quad (21)$$

where $\varphi_i(f)$ is the *i*th DCTC basis vector as in Eq. (11).

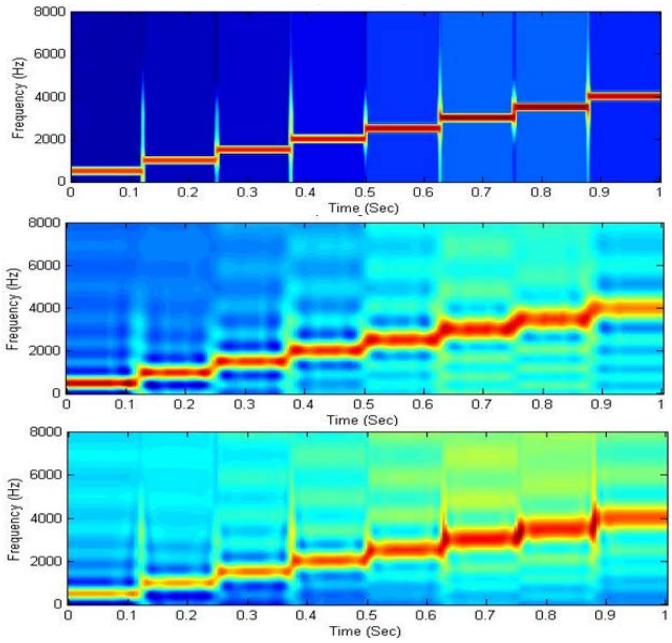

Figure 10: Spectrogram of a signal consisting of a sequence of sinusoids (top), each 125ms long, with a frequency step size of 500Hz. The middle panel rebuilds this signal using uniform time resolution over all frequencies, and the bottom panel rebuilds it with it with increasingly higher time resolution as frequency increases.

In Figure 4, the desired time resolution is plotted as a function of *f*, i.e. DCSCs for low frequencies are computed using low time resolution while DCSCs for high frequencies use higher time resolution. Figure 10 graphically compares the effects of uniform time resolution DCTC/DCS implementation and the frequency-dependent variation. The top panel shows the original spectrogram of a sequence of sinusoids, each 125ms long, with adjacent frequencies separated by 500Hz. The middle spectrogram is a rebuilt version of this signal, using 13 DCTCs and 3 DCSCs with uniform time resolution over all

frequencies. The time resolution within each block is specified by a Kaiser window with $\beta = 5$. The bottom panel presents another reconstructed spectrogram also using 13 DCTCs and 3 DCSCs, but with variable time resolution over frequency. The Kaiser window β values are linearly interpolated between 5 and 15 in this case. In the bottom panel, the time instances at which frequency changes become more clearly marked as frequency increases, which shows increasing time resolution, but in the middle panel, the transitions between adjacent sinusoids are uniform at all frequency boundaries.

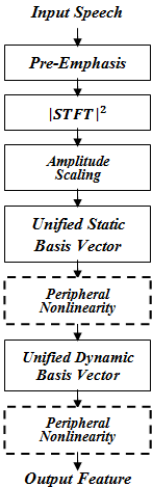

Figure 11: Unified frontend structure

In our previous work [38], we have experimentally shown that in the "standard" MFCC frontend (or other auditory filterbank in place of the Mel filterbank), whose diagram is plotted in Figure 1(a), the nonlinear amplitude scaling can be moved to immediately before the filterbank without degrading the ASR performance. Then, the filterbank weights can be combined with the regular cosine basis vectors (i.e. unwarped basis vectors) by a simple matrix multiplication, which yields the definition of a set of "unified" static basis vectors. Mathematically, suppose the rows of the matrix $\boldsymbol{W}$ contain the filterbank channel response, and the rows of $\boldsymbol{BVF_{reg}}$ contain the regular cosine basis vectors, this unification can be expressed by:

$$\boldsymbol{BVF_{uni} = BVF_{reg} W} \tag{22}$$

In the proposed DCTC/DCS frontend, the unified static basis vectors take the form of a continuous frequency warping $g(f)$, whereas in the MFCC frontend, this warping is implemented by a filterbank. The static features are obtained by a weighted sum of the amplitude-scaled FFT spectrum. Also, we have shown in [38], that the dynamic delta and higher order terms defined in Eq. (1) can also be computed by a summation of the static features, weighted by a set of dynamic basis vectors, in which the nth order basis vector with respect to absolute time is the convolution of all the lower order basis vectors each with respect to its previous order. Thus, a set of unified dynamic basis vectors $\boldsymbol{BVT_{uni}}$ can be defined. The DCS and the delta differential terms are two specific forms of $\boldsymbol{BVT_{uni}}$. With the unified perspective, the final output features $\boldsymbol{F}$ can be rewritten as:

$$\boldsymbol{F} = \boldsymbol{BVT_{uni}} \cdot [\boldsymbol{BVF_{uni}} \cdot a(\boldsymbol{X})]^T \tag{23}$$

where $a(\boldsymbol{X})$ is the amplitude-scaled FFT power spectrum. Figure 11 depicts the block diagram of this unified framework. Eq. (23) reveals the essence of speech features: they can be viewed as a series of linear transformations of the spectrum scaled by an auditory nonlinearity, with optional peripheral nonlinearities in between (dashed blocks in the diagram), such as a family of the sigmoid-shape functions proposed in the work of [39,40], which can improve the noise robustness of frontends. These linear transformations are represented by the unified basis vectors. Filterbanks (or other parts) exert their impact on system quality by shaping the basis vectors implicitly. Thus, the unified basis vectors determine the properties of a frontend. In this sense, the scheme gives us a common "yardstick" to analyze and compare frontends which appear to be different or similar based on the properties of the unified basis vectors.

A high level comparative study can be performed between the standard MFCC frontend (or other type of filterbanks in place of the Mel filterbank, such as gammatone, or the trapezoids in PLP) and the proposed DCTC/DCSC frontend by looking at their unified basis vectors. It's important to notice that though the MFCC frontend and the DCTC/DCSC frontend are based on different logic, they become mathematically identical under the unified framework, except the basis vectors are somewhat different. Figure 12 depicts the first three unified static basis vectors using 26 Mel filters, and the first three unified dynamic basis vectors representing the zeroth order, and delta and acceleration terms. The unified static basis vectors resulting from the Mel filterbank are not as "smooth" as the ones using the continuous Mel-shape warping $g(f)$, which were plotted in Figure 8(a). This is due to the quantization effect. Also, the unified dynamic basis vectors of the differential terms are discrete. Comparing the zeroth order in Figure 12(b) and Figure 8(b), the envelope in Figure 8(b) defines a non-uniform time resolution with the center of the block being emphasized and gradually decreasing towards the ends, while the delta method uses the central term only. This implies that the discrete delta dynamics may not account for the time resolution in encoding the spectral trajectory as well as the proposed DCS basis vectors do.

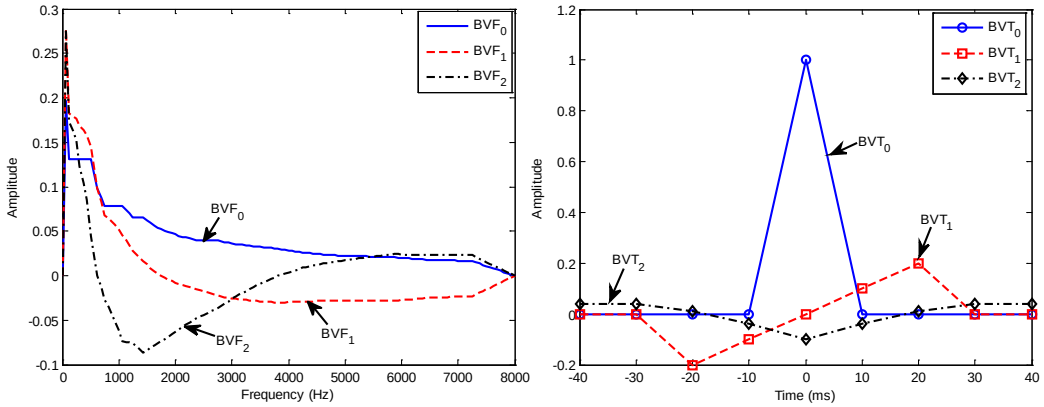

Figure 12: (Left-a) The first three unified static basis vectors resulting from 26 Mel filters, and (right-b) the first three unified dynamic basis vectors from the delta method.

3.4. Experimental Evaluation

A. Experimental Configuration

All the discussion in sections 3.2 and 3.3 involved signal processing techniques for computing spectral features (DCTCs) and spectral evolution features (DCSCs). Although the signal processing for computing these features was motivated by properties of human hearing, the proper way to evaluate the effectiveness of these features for ASR, and to investigate tradeoffs in time and frequency resolution as that affects ASR performance, is to do a comprehensive suite of ASR tests for various conditions and parameter settings. All experiments reported in this section are for phone recognition, with monophone models, using the TIMIT database [41] and HTK ver3.4 HMM recognizer [42]. As is typically done with this database, the entire database, except for SA sentences, was used for either training (462 speakers, 8 sentences/speaker, approximately 189 minutes of training speech) or testing (168 speakers, 8 sentences/speaker, approximately 69 minutes of testing speech). The sampling rate for TIMIT is 16000 samples/second. As is also typically recommended for experiments with this data in [43], the original set of 61 labeled phones were collapsed to 48 phones to create 48 phone models, with a further reduction to 39 phone categories for scoring, as listed in Table 1. Seven groups of similar phones were formed in the reduction from 48 phones to 39 categories: {sil,cl,vcl,epi},{el,l},{en,n}.{sh,zh},{ao,aa},{ih,ix},{ah,ax}. Confusions among phones within each group in testing were not considered errors. The numbers of instances of each of the 39 phone categories are shown in Figure 13.

Table 1: 61 TIMIT phones, and as reduced to 48 for training, and 39 categories (shaded) for testing.

TIMIT Phone	Reduced Phone	TIMIT Phone	Reduced Phone	TIMIT Phone	Reduced Phone
oy	oy	v	v	er axr	er
uh	uh	b	b	iy	iy
aw	aw	f	f	r	r
th	th	w	w	el	el
ch	ch	ey	ey	l	l
y	y	ae	ae	ah	ah
jh	jh	dh	dh	ax-h ax	ax
g	g	d	d	s	s
ng eng	ng	p	p	en	en
sh	sh	eh	eh	n nx	n
zh	zh	m em	m	ih	ih
ow	ow	z	z	ix	ix
hh vv	hh	k	k	#h pau	sil
dx	dx	t	t	pcl tcl kcl qcl	cl
ay	ay	ao	ao	bcl dcl gcl	vcl
uw ux	uw	aa	aa	epi	epi

All HMM acoustic models had three hidden states (plus a non-emitting entry and exit state). A bigram language model was used based on phone bigram frequencies. As is virtually always the case for ASR experiments, primary results for each experiment are given for test data accuracy. However, such results depend not only on the features, but also on the size of the database and complexity of the recognizer. To better estimate the potential for each "best" set of feature parameters, two additional experimental results are given. First, the recognizer is tuned by adding more mixtures, until test accuracy is maximized. This condition is referred to as "BIG_REC." Finally, an accuracy result is given with the original training and test data combined as new training data, but with test results still based on the original test data set. This result could be viewed as an upper

bound on the potential of a particular set of features, if a really large training set were available, implying that the training set represents the test set really well. This result is referred to as "BIG_DATA." It is important to note that results for both BIG_REC and BIG_DATA are obtained with the identical feature settings reported as "best" for each experiment.

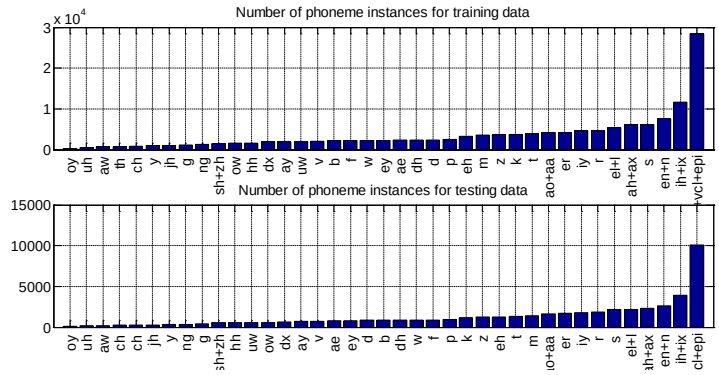

Figure 13: Frequency count of the 39 phone categories in 3696 training and 1344 testing utterances.

In all cases, processing begins with a pre-emphasis using a second order IIR filter:

$$y[n] = x[n] - 0.95x[n-1] + 0.494y[n-1] - 0.64y[n-2] \qquad (24)$$

This second order filter with a peak at approximately 3200Hz is a reasonably good match to the inverse of an equal-loudness contour. In previous work [44], we have found that the complex pole pair results in slightly higher ASR accuracy than the first order pre-emphasis ($y[n] = x[n] - 0.95x[n-1]$). The next step was to segment the signal into overlapping frames, using a Kaiser window with β of 6 (similar to a Hamming window). A 512 point FFT of each frame was computed, and log magnitudes were then computed over the frequency range of 100Hz to 7000Hz. For each frame, the log magnitudes were lower limited to 40dB below the largest magnitude of that frame. In previous work [45], this simple floor was found to improve ASR accuracy slightly for clean speech and more substantially for noisy speech. Thus each sentence was converted to a matrix of spectral values, which were then further processed by the DCTC/DCSC methods presented in Section II & III.

B. Experiment Set 1—DCTC Features Only (Static Features)

For the experiments reported in this section, DCTC features only were computed for each frame. The number of DCTCs was varied (9 to 25 in steps of 2); frame length was varied (5, 10, 15, 20, 25, 30, and 40ms); frame space was varied (2 to 20ms); and the type and degree of frequency warping was varied. Not all combinations of parameter values were evaluated, due to the very large number of combinations which would be required. Rather, most of the parameter values were fixed at what appeared to be the best values, based on pilot experiments, and then a subset of parameter values was varied and performance evaluated. This process was repeated to both examine effects from changes in parameter values and to empirically optimize all parameters for best ASR performance.

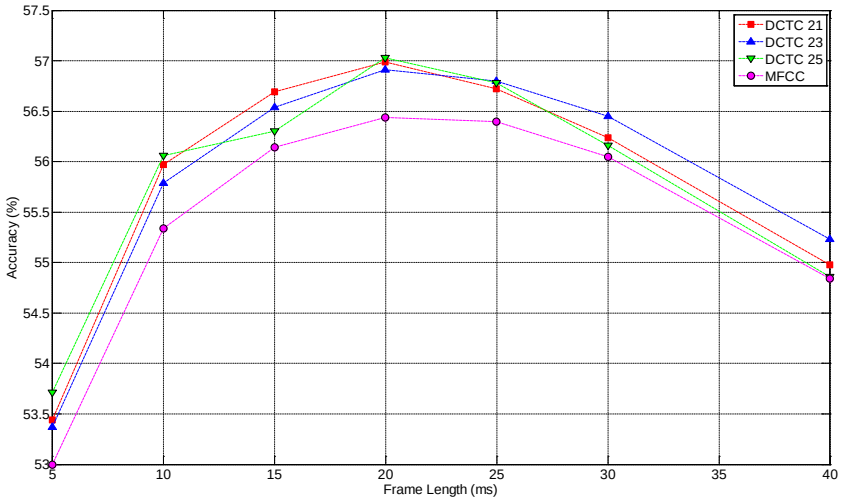

Figure 14: Phonetic recognition accuracy as function of frame length using 21, 23, and 25 DCTCs

Experiment B1—Overall spectral resolution effect for static features: The goal here was to examine effects on ASR performance as a function of overall spectral resolution as determined both by frame length and number of DCTCs used. For these experiments, the frame space was fixed at 8ms, and Mel frequency warping was used. Using 16 mixture HMMs, the number of DCTCs was varied from 9 to 25, and the frame length was varied from 5ms to 40ms. For all combinations tested, ASR accuracy varies from approximately 49% to 57%. However, considering only the range of frame lengths of at least 10ms and at least 13 DCTCs, the range of ASR accuracies is 53% to 57%. Figure 14 plots ASR accuracy using 21, 23, and 25 DCTCs, as a function of frame length. It also contains the plot of accuracy using the MFCC method using 26 filters and 15 DCTCs with frame space fixed at 8ms. The absolute best accuracy (57.0%) was obtained with 20ms frames with spectra encoded with 25 DCTCs. However, the increase in performance for more than 19 DCTCs is minimal, typically less than 0.3%. For the case of static features, frame lengths ranging from 15ms to 30ms results in fairly similar ASR accuracies (difference of less than 0.56% for best result for each frame length). The BIG_REC accuracy is 58.2% (64 mixtures) and BIG_DATA accuracy is 68.5%, thus implying that a high order recognizer trained with a very large data set, using the best parameter values reported here, could improve accuracy at most, by approximately 10%.

Experiment B2—Overall time resolution effect for static features: To examine the effect of the overall time resolution on static speech features, the feature "sampling rate" was varied by varying the frame spacing from 2ms to 20ms. Since the time resolution also depends on the frame length used for FFT calculations, these tests were done with four frame lengths (5, 10, 20, and 30ms). However the number of DCTCs was fixed at 21, and all other parameters were the same as for experiment B1. Results are shown in Figure 15 as compared to the MFCC method using 26 filters and 21 DCTCs with a frame length of 20ms.

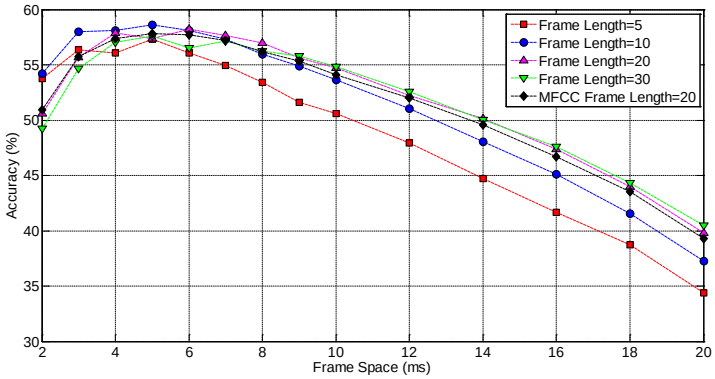

Figure 15: The effect of frame length and frame space on phonetic recognition accuracy for 21 DCTCs

Overall results vary from 34.4% (5ms frames spaced 20ms apart) to 58.2% (10ms frames spaced 5ms apart). ASR accuracy always degrades as the frame space increases beyond certain levels for different frame lengths, with the most severe degradation for the shorter frame length cases. The best performance for each frame length varies over a small range of 57.3% to 58.6%. Not unexpectedly, the best performance is achieved for shorter frame spaces and for the shorter frame length cases. The most surprising result is that accuracy degrades for all frame length cases as the frame space is made very short. Presumably, oversampling of features in time has a deleterious effect on the HMM recognizer, which might be related to the very high correlations of features from frame to frame. The overall best performance of 58.6% was obtained with 10ms frames, spaced 5ms apart. The BIG_REC accuracy is 60.0% (64 mixtures) and BIG_DATA accuracy is 69.7%, thus implying that a high order recognizer trained with a very large data set, for feature conditions reported in this experiment, could improve accuracy by, at most, 11%.

Experiment B3—Effect of frequency warping (auditory frequency resolution) on static features: To test the effect of frequency warping, which determines the auditory resolution of frequency selectivity, bilinear frequency warping was used as implemented in Eq. (7) with a single parameter α to control the degree of warping. Bilinear warping with a coefficient of 0.45 closely approximates Mel warping and a coefficient in the range of 0.5 to 0.57, according to the work in [28] can be used to mimic the Bark warping. Since pilot experiments showed that the effects of frequency warping depend on the number of DCTC features and recognizer order (i.e., number of HMM mixtures), these tests were done for two cases: 13 DCTCs and 8 mixture HMMs, 21 DCTCs and 16 mixture HMMs. In these experiments, 10ms frames, spaced 5ms apart were used. Results are shown in Figure 16, as the warping coefficient varies from 0 (linear warping) to 0.8 (over warped).

The effect of warping is more apparent for the 8 mixture case than for the 16 mixture case (approximately 3% increase in accuracy from linear warping to the best warping with a warping factor value around 0.45 for 8 mixture models versus less than 1% improvement in accuracy for 16 mixture case over the same range of warping). In general, for the larger number of DCTCs and HMM mixtures, the frequency warping has a smaller effect, and best performance is achieved with less warping than Mel (bilinear coefficient of 0.20 versus 0.45). Note that as a control, the normalized version of the

"standard" Mel warping, as proposed by O'Shaughnessy [27] was evaluated, and the result was within 0.1% of the bilinear warping coefficient of 0.45 case, for both 13 DCTCs (8 mixtures) and 21 DCTCs (16 mixtures).

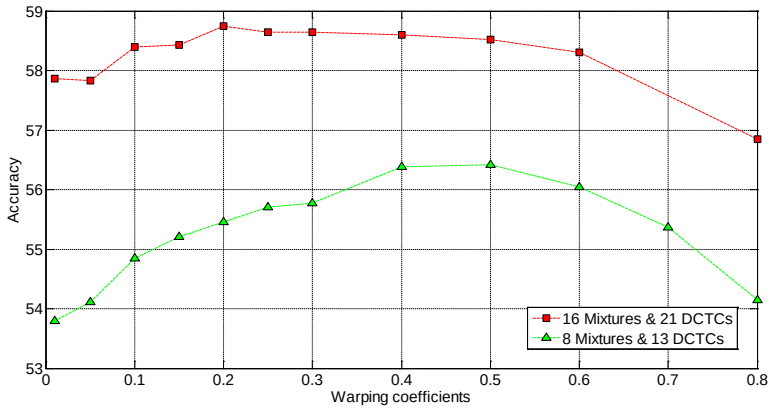

Figure 16: Phonetic recognition accuracy as function of frequency warping for two cases.

The BIG_REC accuracy is 59.0% (64 mixtures) and BIG_DATA accuracy is 68.4%, thus implying that a high order recognizer trained with a very large data set, for the best conditions as reported here, could improve accuracy by at most, approximately 9%.

C. Experiment set 2 —Dynamic features (DCTCs and DCSCs)

In these experiments, a myriad of parameters were varied which were believed to be significant for the case of features (DCTCs/DCSCs), which represent spectral-temporal characteristics in a block of frames centered on each frame. These parameters include number of DCTCs, number of DCSCs, frame length, frame spacing, frequency warping coefficient, block length, block spacing, and time warping coefficient. Not all combinations of parameters were tested, due to both the very large number of cases and low likelihood that some cases would have much effect on ASR accuracy. Based on both pilot experiments and the results reported above, in most of these experiments (C1, C2, C3) many of these parameters were either fixed to a single value, or restricted to a short range. The frequency warping was bilinear with a coefficient of 0.2. The frame length and frame spacing were fixed at 10ms and 1ms, and the block length was fixed at 251 frames (i.e. 251ms block composed of 125ms left and right context plus center frame). The block spacing (which serves as feature spacing to the recognizer) used to control the feature "sampling rate," the time warping coefficient, as well as the number of DCTCs/DCSCs were varied to examine their impacts on the spectral-temporal resolution. The number of HMM mixtures used was 32, due to the large dimensionality of the feature space.

Experiment C1—39 features (13 DCTCs/3 DCSCs) experiments: As a starting point, and also since 39 MFCC features are often used for ASR systems, the first set of experiments was performed with 39 features—13 DCTCs each encoded with 3 DCSCs. First, the effect of block spacing from 4ms to 12ms on ASR accuracy was evaluated, with results depicted in Figure 17. The frame length was fixed at 10ms, and bilinear frequency warping (coefficient of 0.2) was used. The time warping coefficient was 40, using a

Kaiser window. The effect of block space on ASR accuracy is quite small, varying only 1.7% from the lowest accuracy case (12ms block space) to the highest accuracy case (8ms).

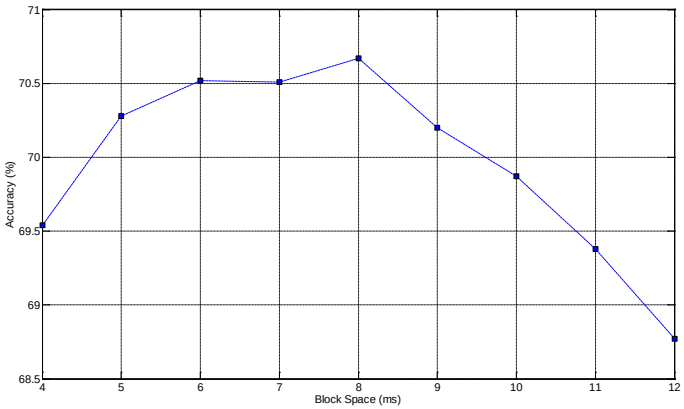

Figure 17: Phonetic recognition accuracy as function of block space, with block length fixed at 251ms.

The BIG_REC accuracy is 71.6% (64 mixtures) and BIG_DATA accuracy is 79.8%, thus implying that for a high order recognizer trained with a very large data set, accuracy could improve, at most, by approximately 9%.

Experiment C2—39 features (13 DCTCs, 3 DCSCs), test of time warping effect (auditory time resolution): In this experiment, the conditions are identical to those for Experiment C1, except that the block length is effectively adjusted by varying the time warping coefficient from 5 to 50 in step of 5, thus creating increasing auditory time resolution. The block spacing is fixed at 8ms, as per the best result in Experiment C1. Results are depicted in Figure 18. The highest accuracy (70.7%) is obtained with a time warping coefficient of 40, but the results do not change much as the time warping is varied from 25 to 50.

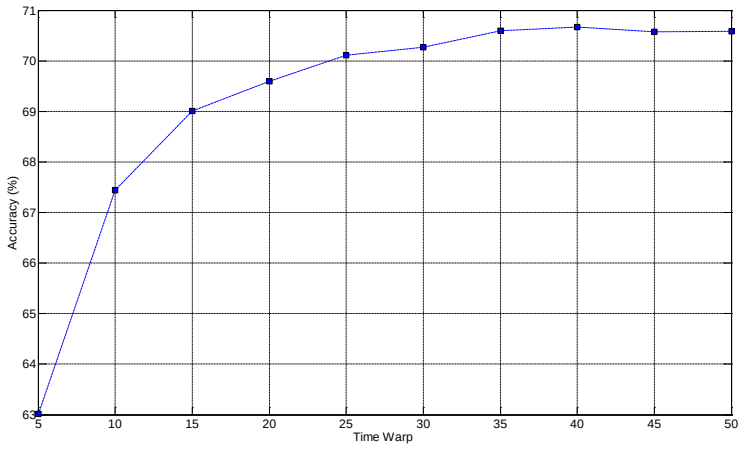

Figure 18: Phonetic recognition accuracy as function of time warping factor for 251 ms blocks, with a fixed block spacing of 8ms.

The BIG_REC and BIG_DATA accuracy for this experiment is identical to that of Experiment C2, implying that for a high order recognizer trained with a very large data set, accuracy could improve, at most, by approximately 9%.

Experiment C3—Overall Spectral-temporal effect: In this experiment, accuracy is evaluated with combinations of the number of DCTCs (9 to 23, in step of 2) and the number of DCSCs (3, 4, 5, 6). These combinations reflect the trade-off between the overall spectral and temporal resolution. Other parameters are fixed as per the "best" settings from previous experiments (frame length of 10ms, frame space of 1ms, bilinear frequency warping with coefficient of 0.2, 251ms block length, 8ms block space, and time warping of 40). Results are shown in Figure 19.

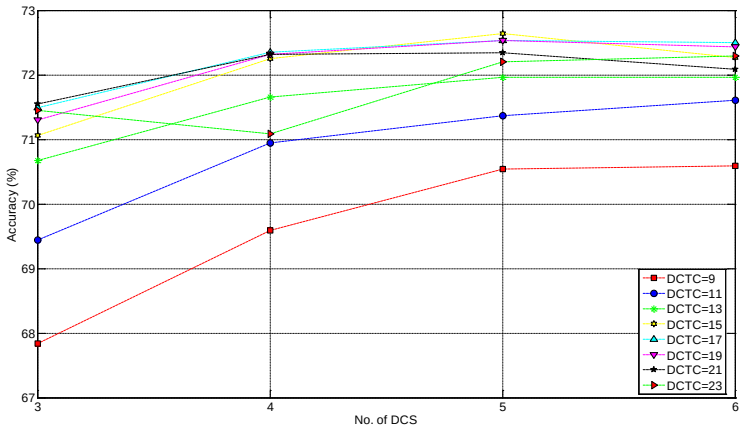

Figure 19: Phonetic recognition accuracy as function of combinations of DCTCs and DCSCs. Other parameters are fixed at their "optimal" values.

For all conditions tested in this experiment, the absolute best accuracy (72.6%) was obtained with 15 DCTCs and 5 DCSCs (75 features). As the number of DCTCs increases beyond about 15, the performance begins to decrease. The number of DCSCs has similar effect. The BIG_REC accuracy is 73.8% (96 mixtures) and BIG_DATA accuracy is 85.7%, thus implying that a high order recognizer trained with a very large data set, could improve accuracy by, at most, approximately 13%.

Experiment C4—"Optimal" parameter values for DCTC/DCSC features-large feature set: Based on the results of Experiment C3, several parameters were further varied. None of the settings tuned improved recognition accuracy by more than .2% above the results reported in Experiment C3. For sake of completeness, the "optimum" settings are listed in Table 2 and accuracies presented in Table 3. Note that "optimum" is only with respect to settings evaluated. However, given the relative insensitivity of ASR accuracy to the settings varied, it does seem unlikely that any other settings, within the framework described in this section, would results in substantially higher ASR accuracy.

Table 2: "Optimum" parameter settings for large feature set

Parameter	Value
Frame Length	8 ms
Frame Spacing	1 ms
FFT Length	512
Frame Window	Kaiser window, $\beta = 6$
Frequency Warping	Bilinear, $\alpha = 0.2$
Number of DCTCs	15
Number of DCSCs	5
Frames per Block	250 frames
Block Spacing	7ms
Time Warping	Kaiser window, $\beta = 40$

Table 3: "Optimum" 75 feature ASR accuracies

Condition	Number of HMM mixtures	Accuracy (%)
Regular test	32	72.8%
BIG_REC	96	74.0%
BIG_DATA	96	85.2%

Experiment C5—"Optimal" parameter values for DCTC/DCSC features-small feature set: The best results reported in the previous section were based on a large number of features (75). Based on the rationale that "optimum" parameter settings for a much smaller feature set, in this experiment, "optimum" parameter settings were experimentally determined for a small feature set (27 features). The values are given in Table 4 and accuracies listed in Table 5. As before, "optimum" is only with respect to settings evaluated.

Table 4: "Optimum" parameter settings for small feature set

Parameter	Value
Frame Length	8 ms
Frame Spacing	1 ms
FFT Length	512
Frame Window	Kaiser window, $\beta = 6$
Frequency Warping	Bilinear, $\alpha = 0.45$
Number of DCTCs	9
Number of DCSCs	3
Frames per Block	250 frames
Block Spacing	7ms
Time Warping	Kaiser window, $\beta = 50$

Table 5: "Optimum" 27 feature ASR accuracies

Condition	Number of HMM mixtures	Accuracy (%)
Regular test	16	69.3%
BIG_REC	64	69.0%
BIG_DATA	64	77.8%

3.5. Conclusion

This section presented a generalized spectral-temporal feature extraction frontend for representing speech information. The feature set is motivated by the attempt to mimic two primary properties of human hearing: frequency and time resolution. Based on frequency warping and frequency-dependent time warping built into modified 2-D cosine basis vectors, the relative importance of spectral and temporal features could easily be evaluated, and the trade-off between auditory frequency selectivity and time resolution was explored. A wide range of ASR experiments was conducted based on the DCTC/DCSC implementation of the proposed framework to comprehensively study the spectral-temporal resolution effects of human hearing by tuning the parameters of the frontend towards either side of the spectral-temporal trade-off, and the "optimum" combining point was found.

In addition to the DCTC/DCSC method, which uses uniform time warping for all frequencies, we also implemented the DCSC/DCTC variation, which incorporates frequency-dependent time warping (experimental results not presented). Specifically, we used the best warping factors obtained in the DCTC/DCS experiments (i.e. 50 for the 27 feature case and 40 for the 75 feature case) as the baseline, and imposed smaller time warping for lower frequencies compared to higher frequencies, with averages fixed at the baseline values. The results only showed a minor effect on the recognition accuracy. This seems to imply that, although this auditory effect has been verified by neurophysiological and psychoacoustic studies [30,31,32], it does not play a crucial role, at least for the phone recognition ASR task evaluated in this section. Based on the ground work of the proposed frequency-dependent DCSC/DCTC scheme, which provides a tool to explore the auditory time-frequency trade-off, it still remains an open area for future research to further study whether this frequency-dependency is important for various ASR tasks, and if it is, how to more effectively incorporate this theory into an ASR frontend.

Another possible direction in the future work involves the idea of using a non-symmetric window for the time resolution, i.e. the *dh/dt* term in the DCS basis vectors. Acoustic evidence found in [46] shows that the left context is more informative than the right context in phoneme recognition. This finding implies that a non-symmetric window, tilted toward the left side of the center point for a time block, thus assigning higher time resolution to the left context, might be beneficial.

4. AUTOMATIC WORD TO MORPHEME DECOMPOSER FOR RUSSIAN

4.1. Introduction

The Russian language is a synthetic language. That is, words are constructed from basic building blocks called morphemes (prefix, root, suffix and inflection) [47]. The same morphemes can be used to form new and different words. A prefix, root and suffix make up the stem of the word. For example, the word 'подходящий,' translated to 'appropriate' shown in Table 6, consists of the prefix 'под', root 'ход', and suffix 'ящ'. These morphemes make up the stem plus the inflection 'ий'. The stem may contain just one morpheme, that is the root, or several morphemes, including several prefixes, suffixes, and roots.

Russian words are on average longer than words of other languages, and exhibit clearer morphological patterns [2]. As a result, longer fragments of speech must be analyzed during the speech recognition process.

Table 6: Decomposition of a word into morphemes.

<table>
<tr><th colspan="4">Подходящий</th></tr>
<tr><th>Под</th><th>ход</th><th>Ящ</th><th>ий</th></tr>
<tr><td>Prefix</td><td>root</td><td>Suffix</td><td rowspan="2">inflection</td></tr>
<tr><td colspan="3">Stem</td></tr>
</table>

The Russian vocabulary contains more than 160 thousand lemmas, or words in canonical, or dictionary, form [49]. Usually, the inflections of the word will change, while the stem of the word remains the same. As a result, there are a high number of conjugated (inflected) word forms with different inflections but only one stem. For example, the verb "делать" or "to do" has over 100 word forms.

Noun inflections identify number (singular or plural) and can be conjugated to six different cases. Similarly, verb inflections identify gender, number, tense, voice, aspect etc. Additionally, pronouns, adjectives and numerals can be conjugated with different inflections [47]. Consequently, the number of unique word forms to be recognized by a Russian ASR system increases to over 3.7 million [50].

Another peculiarity of Russian is that sentences are not defined by a strict set of grammatical construction rules because inflections convey all the grammatical meaning within a sentence [51]. Changing the word order within a sentence does not alter the meaning. However, depending on text style, some word orderings are preferred stylistically over others.

The specifics of the Russian language, like longer words, relaxed ordering of words within a sentence, and the amount of unique words, decrease the performance of conventional ASR systems that use traditional approaches, such as n-grams, for the statistical language model [52]. Instead, morpheme level representation of speech is

proposed for use in automatic speech recognition systems. Use of morphemes decreases the size of the vocabulary of base lexical units by several orders and increases the speed of Russian ASR systems as will be explained below.

4.2. Morpheme Database

The database of prefixes, suffixes and inflections was created using the Vocabulary of Morphemes of the Russian Language by Kuznetsova and few other published dictionaries [53-56]. The root database was kindly provided by Dr. Kaprov of the St. Petersburg Institute for Informatics and Automation of Russian Academy of Sciences. The total number of morphemes to be recognized by the ASR system is a little over 17,000, as shown in Table 7. This is a drastic reduction from over 3.7 million unique words that need to be recognized by the Russian ASR system.

Table 7: Morpheme Database.

Prefix	79
Root	16776
Suffix	478
Inflection	49

4.3. Background

There has been much research done on word and stem level Russian ASR systems that are speaker independent and large vocabulary, such as the trigram statistical model, developed by IBM [57]. This system was trained by 30,000 utterances. The trigram LM was trained on 40 million word textual data. A system of Russian phonetic sub-groups and a set of rules for phonetic transcription of words were developed. This system attained 5% word error rate (WER) but it was not further developed due to inflective nature of Russian, its huge vocabulary, and the rigid word ordering of words within a sentence.

In 1999 – 2001, a collaborative project by Intel and All-Russian Research Institute of Experimental Physics – Software Technology Laboratory (VNIIEF-STL) resulted in a large vocabulary Speech Developer Toolkit (STD) [58]. The toolkit involves modules for vector calculation, construction and adaptation of acoustic models, speech decoding by finite state and stochastic grammars etc. Currently, VNIIEF-STL is developing a Russian ASR system with a vocabulary of over one million words.

There is only one commercially available Russian ASR system today. The line of "Gorynich" systems was developed by "VoiceLock" on the basis of the Dragon system. This system's recognition accuracy is ~70%.

Speech Informatics Group of St. Petersburg Institute for Informatics and Automation of the Russian Academy of Sciences (SPIIRAS) introduced a morpheme level Russian ASR system in 2005 [59, 63]. The morpheme database of over 17,000 morphemes was employed which greatly reduced the vocabulary of recognizable lexical units and

increased the machine processing speed. This system attained 90% recognition accuracy with a very small vocabulary of 1850 words.

Overall, in the past decade attempts by companies like Intel and IBM to create speaker independent Russian ASR systems for large vocabularies were unsuccessful. The Russian ASR system developments presented were either halted for economic reasons, did not have large vocabularies, or did not yet attain significant WER improvements [47].

The motivation for the present work is to provide an open source tool for a Russian word to morpheme decomposer. Such a tool is needed as a first step in implementing morpheme level LM, which could be an important part of an overall Russian ASR system, due to the structure of Russian, as mentioned above. The intention is to combine the morpheme level with root and/or word level LMs to improve accuracy for Russian ASR systems. The goal for the ASR system is speaker independence and large vocabulary. Its simplicity of implementation and accuracy of decomposition makes it a great tool to have. The size of the program with the prefix, suffix and inflection databases is only 44 KB.

In contrast, there are two tools available online for word to morpheme decomposition [55, 56]. These tools have several issues associated with them: these are commercial tools, they are not available in source code format, and there are no publications or documentation associated with them available to the public. Furthermore, the method presented here is simple, accurate and only uses the databases of prefixes, suffixes and inflections and not the biggest – root database. The work by Aleksey Karpov of SPIIRAS uses all four databases and has a very sophisticated algorithm that runs through each morpheme database several times for every word decomposition [63]. This makes the program slow and difficult to implement. Similarly, the work by Edward Whittaker of the University of Cambridge [52, 62] uses only 28 prefixes and sixty suffixes for decomposition of a word. The prefixes and suffixes were then separated systematically from the word beginnings and ends, respectively, using a simple string matching operation. All words were eligible for decomposition and the two morphemes were separated wherever a match was made, irrespective of whether the match was linguistically correct or not. Consequently, this resulted in numerous incorrect decompositions.

4.4. Data preparation

All textual data is preprocessed to fit the requirements of the HTK toolkit [60]. The format of data is one word per line, all punctuation is removed, and there are no capital letters and numbers. For example, the final format of the word ‘перестройка’, or ‘restructuring,’ is shown in Figure 20. Start symbol, <s>, is followed by white space, then Russian word, followed by a white space and end symbol, </s>. 1500 words of data were used for initial development and testing, and 20,000 words of data were used for more thorough testing.

<s> перестройка </s>

Figure 20: Data Format

A special program was written to convert numbers to words. This is needed since the convention in Russian (as well as most other languages) is to write numbers as "numbers" but speak them as words. This program takes care of conjugation of numerals within a sentence to convey correct grammatical meaning. For example, the numeral "3rd" or "третий" has 29 word forms.

Consider the conversion example for the number 2,324,015. First, the number is divided by a million, and the correct conjugated form of the word 'million' and digit are determined. Here, *digit.m* is accessed to arrive at two or "**два**," and *million.m* to arrive at a million or "**миллиона**." Second, since thousands are in hundreds, the *hundred.m* function is accessed. Here, proper word for 300 is "**триста**." Then, *ten.m* is accessed for 20 which is "**двадцать**." It is followed by *digit.m* which results in "**четыре**." For thousands, the proper term is "**тысячи**" and attained via *thousand.m*. The last term fifteen or "**пятнадцать**" is obtained from *teen.m*. The final term of the number in words is "два миллиона триста двадцать четыре тысячи пятнадцать."

Table 8: Number to word conversion of 2,324,015.

digit.m	Два	Четыре
ten.m	двадцать	
hundred.m	Триста	
teen.m	пятнадцать	
thousand.m	Тысячи	
million.m	миллиона	

4.5. Decomposition algorithm

The first effort made to write a word to morpheme decomposition program was coded in C++. The word decomposition error (WDE) for this initial code was 5% and there was no clear way to reduce this WDE. After much deliberation, a completely new approach was taken, and implemented in MATLAB, due to the much higher level of built in useful functions for the decomposer. Overall, the new decomposition algorithm, described in this section, is greatly simplified from the first version (and much more accurate). This algorithm uses the prefix, suffix, and inflection databases only. The root database of 16776 morphemes is not employed. The textual data processing time is greatly decreased and, most importantly, the WDE is decreased to less than 0.2%. The accuracy was examined manually and compared to online server tools [55, 56].

The algorithm first looks for the start symbol and searches for the prefix match. Once the prefix matches, the morpheme is separated from the rest of the word by a delimiter. Next, the algorithm looks for the end symbol and searches for both the suffix and inflection matches. Delimiters are put before and after the suffix once both morphemes are matched. For each suffix there are only a certain number of inflections possible, which

can range from zero to 49 inflections. In Table 9, suffix 'k' has 32 possible inflections. All possible inflections for each of 478 suffixes have been compiled.

Table 9: Possible inflections for suffix 'к'.

Suffix	Possible inflections
К	ый, ий, ой, ей, ое, ее, ые, ие, ого, его, ому, ему, ая, яя, их, ых, ах, ыми, ими, ами, ым, им, ом, ем, ою, ею, ую, юю, у, о, **а**, и

The decomposition of the word 'перестройка' using this algorithm is shown in Table 10. By first looking at the start symbol, the prefix 'пере' is found and a delimiter '/' is placed after it. Next, the end symbol is found and the suffix '**к**' plus the 31st ending '**а**' are matched, and a delimiter is placed before and after the suffix. The root of the word "**строй**" is separated as a result. Morphemes are not labeled during the decomposition process as this does not aid the morpheme recognition step.

Table 10: The decomposition of word 'перестройка'.

function	morpheme	'<s> перестройка </s>'
prefix.m	'**пере**'	'<s> **пере**/стройка </s>'
suffix.m	'**к**(**а**/>)'	'<s> **пере/строй/к/а** </s>'

An algorithm for automatically parsing speech and breaking it into parts almost never has 100% accuracy. The decomposition algorithm presented here is not an exception to this unfortunate reality. There are three main exception categories with this algorithm: two-root words, over-decomposition, and root-suffix overlap.

4.5.1. Exceptions to the Algorithm – Two-root Words

There are over one hundred words composed from two roots connected directly without a morpheme [case 1], by a suffix [case 2], a combination of suffix and prefix [case 3], or a combination of two suffixes [case 4], as shown in Table 11. Roots are shown in bold. A database of two-root words has been compiled and included within a *prefix.m* function as a special case handle.

Table 11: Sample of two-root words.

1	**Авиа**/**мотор**/н/ый	2	**Вод**/о/**лаз**
3	**Лес**/о/за/**готов**/к/а	4	**Даль**/н/е/**восточ**/н/ый

4.5.2. Exceptions to the Algorithm – Over-decomposition

In Russian, there are over 470 suffixes. Certain short suffixes may match parts of longer suffixes, which may result in longer suffixes being incorrectly decomposed if there is a possible inflection match. Additionally, a long suffix may be decomposed once more by a shorter suffix which results in over-decomposition. A solution was introduced to avoid this complication. All suffixes that had matches in parts were put in one line of code, as shown in Figure 21. Once there is a long suffix match shorter suffixes are not even considered for decomposition.

ов**ищ** ⇨ л**ищ** ⇨ б**ищ** ⇨ **ищ**

Figure 21: Longer suffixes come first.

4.5.3. Exceptions to the Algorithm – Root-suffix Overlap

In a few rare instances parts of the root and suffix may overlap, resulting in incorrect decomposition. A database of root-suffix overlaps has been compiled and included within the *suffix.m* function as a special handle. The root '**голов**' for the Russian word '**головизна**' overlaps with the suffix '**овизн**,' resulting in incorrect decomposition. To avoid this issue the word '**головизна**' along with another 45 exceptions are included within the *suffix.m* function as a special handle cases.

4.6. Experimental results

The program was trained on a textual data of 1500 words obtained from [54] and achieved 100% accuracy. The WDE of 0.2% was obtained for the test data of approximately 20,000 words from the Russian open-source YouTube database [61]. The textual data used for the experiment was of conversational format. The accuracy was examined manually and compared to online server tools [55, 56].

4.7. Conclusion

Today, only two online server decomposers are available for commercial use. In contrast, the off-line, compact decomposer is in the public domain, free of charge, and intended to further the research in the Russian ASR systems area. The WDE rate of the tool is very low and makes it very reliable. Subsequently, this tool will be used in building a morpheme level LM, followed by a morpheme level acoustic model. These are intended to be used along the word level language and acoustic models within the Russian ASR system to improve on recognition accuracy.

5. NON-UNIFORM FRAME SPACING FOR SPEECH FEATURE CALCULATIONS

5.1. Introduction

The primary objective of the work reported in this section of the report is to investigate a number of algorithmic (i.e. automatic) methods for non-uniform time sampling of speech features and to determine the merits of these methods for improving the accuracy of

automatic speech recognition. Since the accuracy of an ASR system using fixed frame spacing depends on what that spacing is, an important constraint used in this work is to consider only methods for variable frame spacing such that the average frame spacing is the same for each sentence long utterance. That is, the total number of frames for each utterance is the same for the fixed spacing approach (the control) and the variable spacing method.

The general strategy used is to first develop a number of mathematical methods for controlling the frame spacing, subject to the constraint mentioned above, and then to test these methods with ASR experiments. All of the experimental testing was done using the TIMIT acoustic-phonetic database (widely used in the ASR community) and using a powerful Hidden Markov Model Toolkit for the recognizer. All experimental results are in terms of phonetic recognition accuracy. Experimental tests were done primarily with clean speech, and a limited number with white noise added to clean speech.

Three algorithms were considered: L1-Norm frame deletion, Delta Coefficients frame deletion, and non-uniform regression analysis. Only the L1-Norm frame deletion and non-uniform regression analysis are considered here, both theoretically and experimentally.

The idea of non-uniform frame spacing stems from the theory that consonant sounds are not well represented by a fixed frame spacing. In other words, even though it is typically assumed that a speech signal is stationary over a time period of approximately 10-20ms, and adequately modeled using 25ms overlapping frames spaced apart 8ms, literature has shown that this assumption is not always valid. In order to solve this problem, the adoption of variable frame spacing seems like a plausible approach. That is, the frame spacing for consonants (or rapidly varying spectral regions) can be made to be much shorter than in the vowel regions (slowly varying spectral regions). Thus, as a function of frame index, the rapidly changing portions of a speech signal are lengthened to increase their weight in the feature extraction stage of ASR. There are a few ways to approach this problem. One way is to oversample a speech signal to the point where consonants have sufficient representation. The problem with this approach is that the computational load becomes burdensome since the vowel and other slowly changing portions of a speech signal are unnecessarily over sampled. The chance of insertion errors also increases with this approach. Insertion errors occur when the ASR system recognizes extra phonemes that are not really present. Furthermore, as shown later, this oversampling actually can degrade ASR accuracy.

Another way to approach the problem is to create a deletion criterion that removes frames of a speech signal that vary slowly over time. The key to such an approach becomes the chosen deletion criterion. The measure of speech variation is not an exact science since no two people speak the same way, so difficulty arises in developing a threshold for frame deletion. In this work, three methods of frame deletion were examined, as mentioned previously.

One basic assumption in the theoretical development and experimental work reported in this thesis is that the average frame rate is unchanged. As demonstrated experimentally in a later section, HMM ASR system accuracy does depend on the frame spacing, for the typical fixed frame rate approach. And, interestingly, the overall best frame spacing is not the shortest one, but some intermediate value of about 8 ms. To avoid the possible confounding effects of changing the overall frames rate, in our work, we only considered variable frame rates where the average frame rate, in each sentence, was unchanged from the control fixed frame rate. The basic approach is to begin by oversampling and then to delete and /or resample features to match the control fixed frame rate case. This "resampling" is done using the interp1 function in Matlab, which is a 1-D linear interpolation procedure. Although the interp1 function was used with the default linear method, other interpolation methods such as cubic or spline could be implemented.

5.2. L1-Norm Frame Deletion

The method of measuring spectral change using L1-Norm between a frame and the subsequent frame is explained in this section. Shown below is the spectrogram of a three second utterance of the sentence "Don't ask me to carry an oily rag like that," spoken by a female speaker.

In this spectrogram there are 1810 frames, each represented by 189 frequency values (100 Hz to 7000 Hz). This spectrogram is essentially a 189x1810 (mxn) matrix of amplitude values at varying frequency and time locations. This particular spectrogram was generated with a frame space of 2 ms. The L1-Norm method of frame deletion follows a fairly simple initial procedure:

1) Each frame (column vector of size m x 1) of the spectrogram is subtracted from the next frame which gives an m x n-1 matrix.
2) The absolute value of each column is then taken so that all differences are positive
3) All values in each new column are summed, to produce a 1 x n-1 vector. This 1 x n-1 vector is the L1-Norm measure of spectral change. A plot of this vector is figured below for the corresponding spectrogram

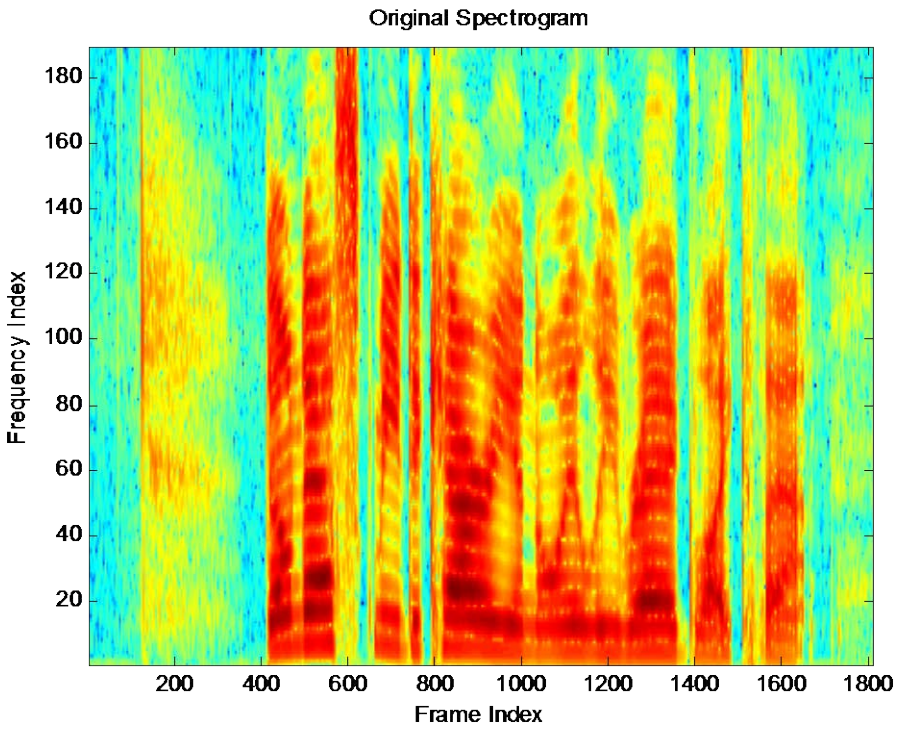

Figure 22: Original Spectrogram of Example Utterance

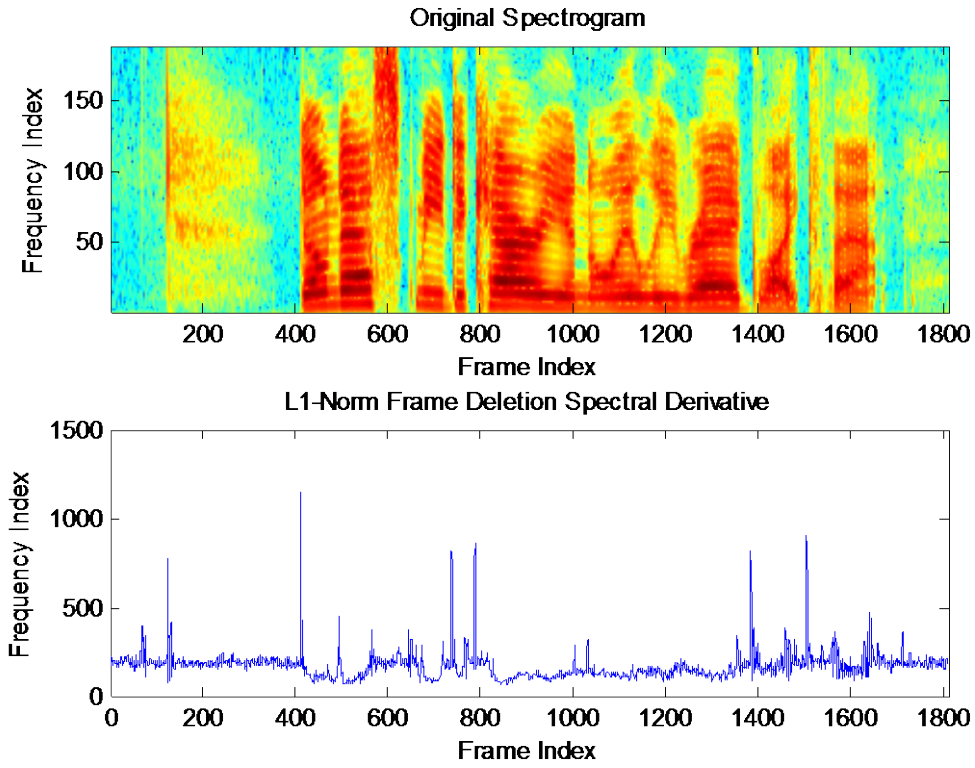

Figure 23: Original Spectrogram (Top)/ L1-Norm Frame Deletion Spectral Derivative Vector Plot (Bottom)

At a glance the variation between the noise and speech portions of the spectrogram is almost indiscernible. The only acceptable parts of the spectral derivative plot are the spikes which show the beginning or ending of spoken speech. The rest of the plot does not do a good job in differentiating speech and non-speech, or consonant and vowel portions of the spectrogram. To address this problem, the energy of each individual frame is taken into account, and used as a weight to put more emphasis on spoken speech and less emphasis on noise. The way this is done is by multiplying the spectral derivative value by the average amplitude in the current frame. An equation to show how this is done is given below.

$$D(i) = D(i) * \frac{\sum Frame(i)}{\max(FrequencyIndex)} \tag{25}$$

"Σ" in the equation means sum all values. This equation basically scales the spectral derivative by the "energy," loosely speaking, of the current frame. The maximum of the frequency index is 189, which is the total number of values you would sum for each Frame(i). The measure of spectral change, after doing this extra step, is given in the figure below.

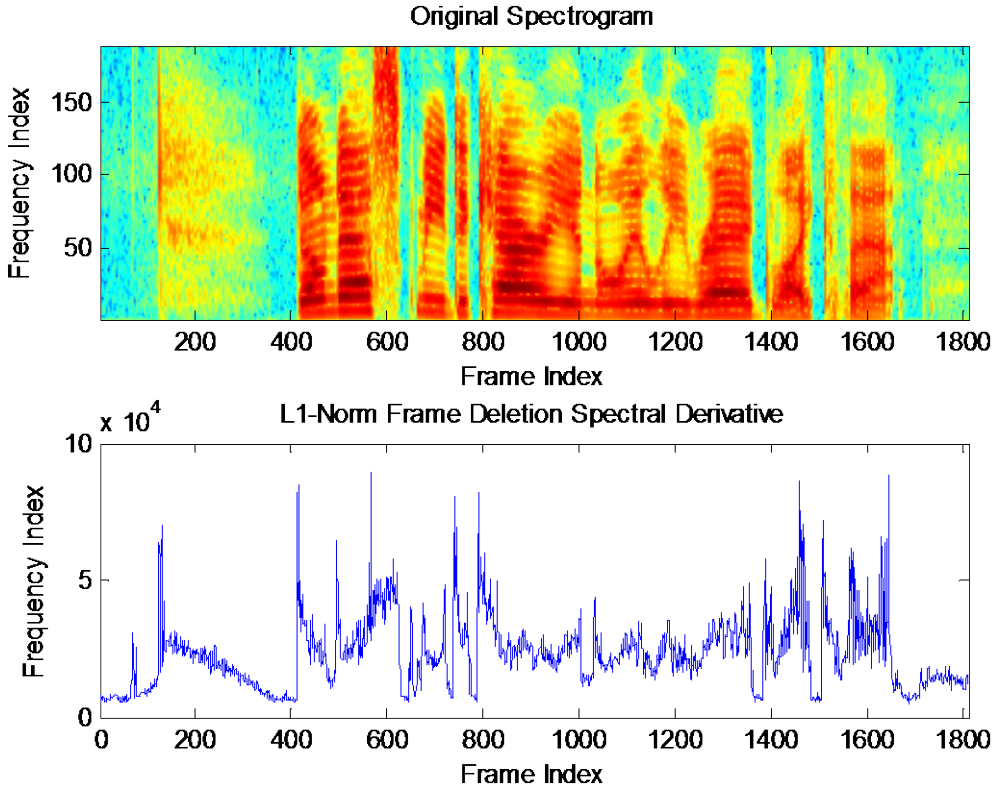

Figure 24: Original Spectrogram (Top)/ L1-Norm Spectral Derivative Weighted by Frame Energy

Now, the noisy and slowly changing regions (vowels) have lower measures of spectral change. This appears to be much better, graphically speaking, than the previous plot of spectral variation. However, there still remains the problem of selecting a sufficient threshold criterion. Large swaths of successive frames can be discarded if the threshold

for deletion is too low. Conversely, an insignificant amount of frames would be removed if the threshold for deletion is too high. In order to account for this problem, an additional check was implemented in the algorithm for selecting which frames are to be deleted. A check is placed on the maximum number of frames "allowed" to be deleted in succession, regardless of the spectral variation. Experimentation shows that setting this check to four frames gives the best performance. That is, for the cases tested, removing more than four frames in succession does more harm than good in the overall algorithm for non-uniform frame spacing. The last piece of the algorithm that needs to be determined is the actual threshold criterion. There are two general ways to select a threshold criterion. One way is a global threshold based on the average spectral variation of a speech database. The other way is a local threshold based on the variation in each individual utterance. For the experiments reported here, the latter approach was used. For each individual utterance in the database, a percentage of the average variation was used as the threshold. Different percentages were tested to obtain the optimal level (empirically speaking). After the frames were discarded, one final step was taken to ensure that the time information remains sufficient in the recognition portion of ASR. That step was to re-interpolate to the original number of frames. The figure below gives the original spectrogram, the spectrogram after deleting frames based on a 50% threshold, and the final spectrogram after re-interpolating to the original 1810 frames.

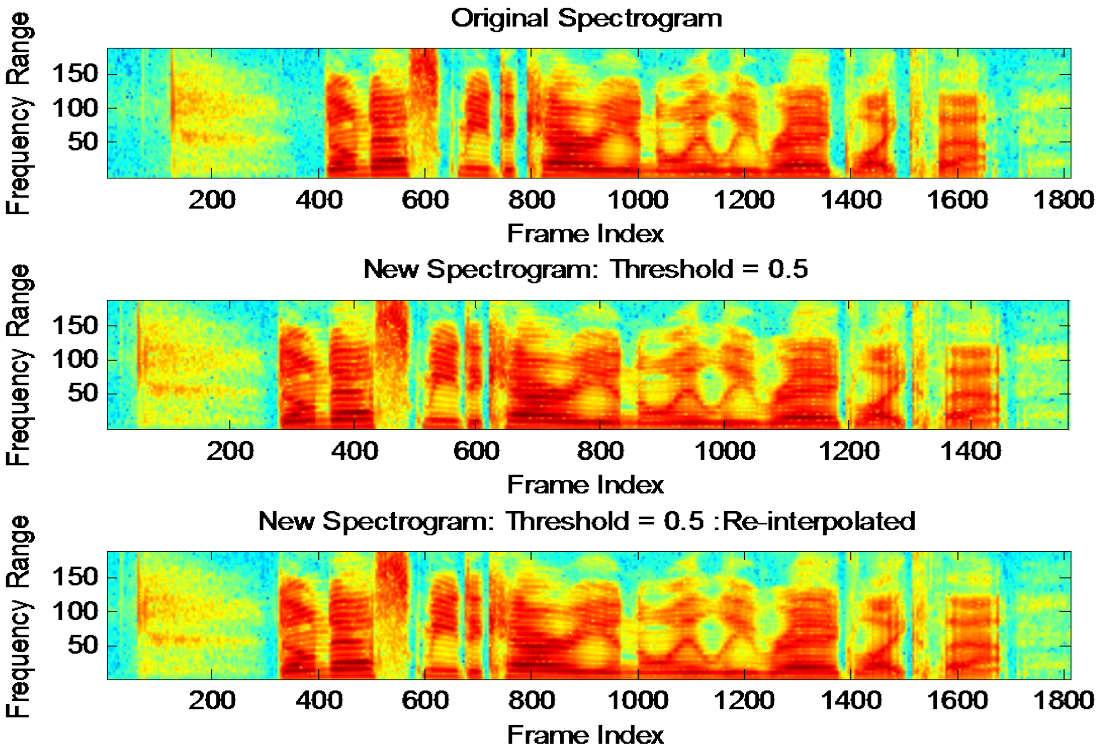

Figure 25: Original Spectrogram (Top)/ L1-Norm Frame Deletion for 50% Threshold (Middle)/ Reinterpolated Spectrogram (Bottom)

The next step was to establish the criteria for frame deletion. In addition to choosing a threshold for deletion based on the average variation of the utterance, a check on the

number of frames to be deleted in succession was also implemented. As in the L1-Norm frame deletion algorithm, deletion of more than four frames in succession was found to be more detrimental than beneficial to the overall recognition performance. The algorithm thus included a four frame check to prevent this from occurring.

5.3. Non-Uniform Regression Analysis

A recursive approach to reshaping the spectrogram was also developed. To explain this method, this section of this thesis provides a description and simplified example of the algorithm used. The ultimate goal remains the same--non-uniformly readjust the frame space to account for variability in the spectrogram, while keeping the average frame spacing in each utterance (total number of frames per utterance) fixed. A primary difference between the method presented in this section, and the methods of the previous two sections, is that the constraint of keeping the average frame spacing fixed is fundamentally integrated into this method, rather than requiring a final interpolation, as needed in the previous two methods. However, the method of this section still critically depends on a measure of spectral change.

To begin, let us assume we have a spectrogram from which we arbitrarily generate a function, d, of spectral change. In other words, for rapidly changing portions of the spectrogram, d values are higher. Since reshaping the spectrogram according to a variable frame space was the goal, the inverse of d is computed to provide a non-uniform frame spacing function, g. For large variations in the spectrogram, there are small values of g (small frame spacing). This g function can be linearly scaled (ax+b type scaling) to a specified maximum and minimum value for g (g_{max} and g_{min}). Since the intent is to use g to scale the frame space T, we can also predefine the bounds of the frame space between T_{min} and T_{max}. The reason for this is to prevent scenarios where g may be so small or large that it would result in really small or really long frame spaces. Assuming an original frame space of 2ms, reasonable values for T_{min} and T_{max} might be 1ms and 3ms respectively. We now need to find parameters that will shape g using the predefined values for T_{max} and T_{min} using the system of linear equations below.

$$T_{max} = a * g_{max} + b \quad (26)$$
$$T_{min} = a * g_{min} + b \quad (27)$$

With two equations and two unknowns, we easily solve for a and b. Now the new frame space function, g_{new}, is given by the following equation.

$$g_{new} = a * g + b \quad (28)$$

This is the general idea for reshaping the g function. As an example, let us assume we have a g function with 100 values generated from 100 frames of a spectrogram. If there was an initial fixed frame space of 2ms, then the total time length of all frames would be 200ms. Now, we also assume constraints T_{min} and T_{max} of 1ms and 3ms. The ratio of T_{max} to T_{min} is called the scaling factor, c. Let's say after calculating the g function we arrive at three different levels of variation. The levels are small frame space, medium

frame space, and large frame space (large variation, medium variation and large variation, respectively). If 25 g function values are small, 50 are medium, and 25 are large, we can use this to recreate the g function knowing that the total time must still add up to 200ms, and that the values must remain within the T_{min} and T_{max}constraints. We then recursively hone in on the ideal number of small, medium, and large frames with different frame space values that retain the parameter constraints. The ultimate goal is to keep the total sum of frame space values roughly equal to the original sum. In the algorithm, the original sum would have been scaled to 1 second, so the new sum must also be roughly equal to 1 second. Ideally it would be exactly 1 second, but for real signals there is an error. Consequently creating an error threshold would allow for the algorithm to perform its task of producing the new vector of frame space values. The error threshold used was 0.002 (0.2%). Plots of the original spectrogram along with the spectrograms after the algorithm (with scaling factors c=2.5, 5, 7.5, and 10) are shown in the figures below.

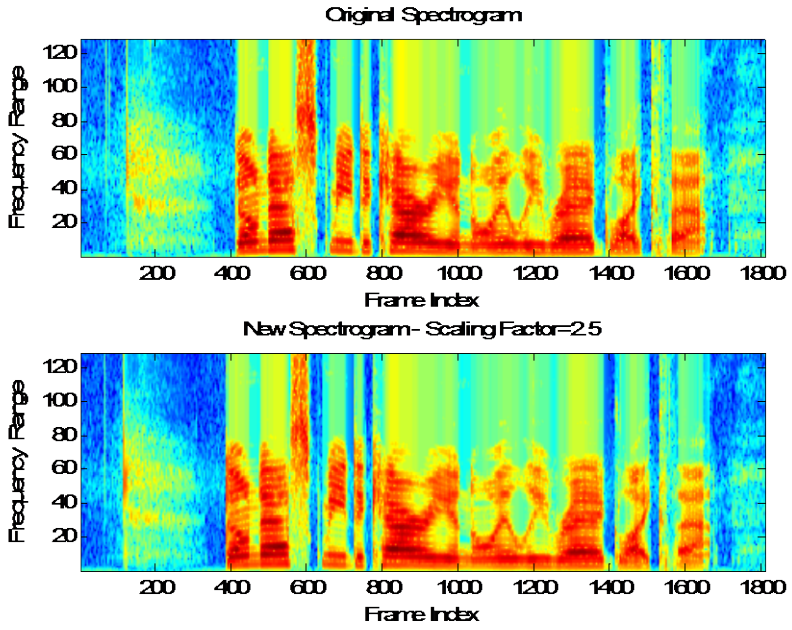

Figure 26: Regression Analysis (Scaling Factor=2.5)

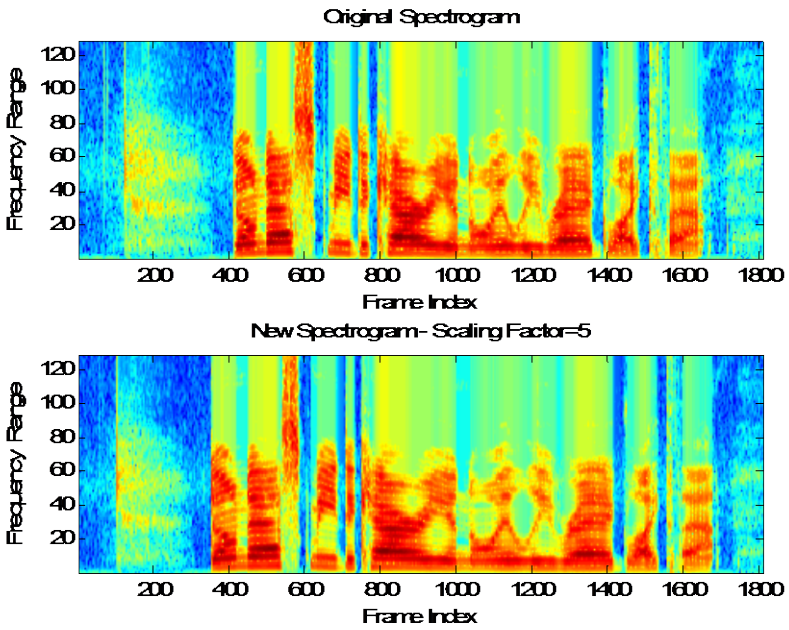

Figure 27: Regression Analysis (Scaling Factor=5)

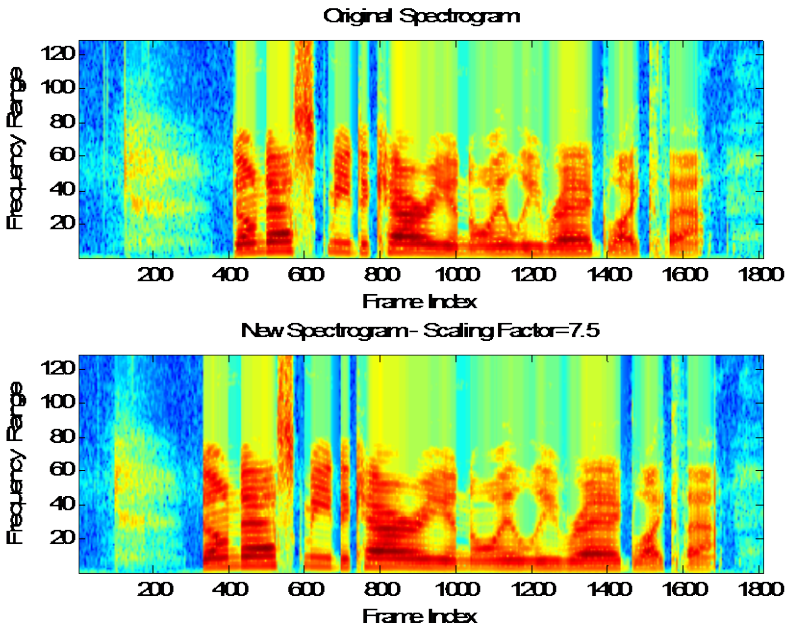

Figure 28: Regression Analysis (Scaling Factor=7.5)

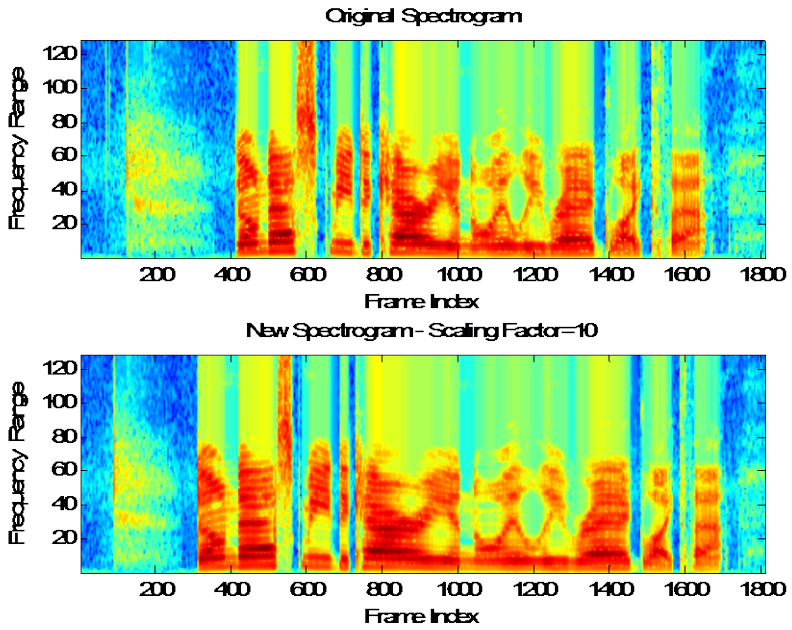

Figure 29: Regression Analysis (Scaling Factor=10)

As can be observed from these figures, the noisy regions seem to slowly disappear whereas the faster changing and speech portions stretch as the scaling factor increases. However, upon closer examination, it can be seen that that vowel regions also stretch due to the fact that they change faster than the noisy and silent regions. This is one major drawback to the regression analysis algorithm, because ideally we would want the vowel regions to shrink just as much as the noisy and silent regions. Because of this drawback, the current algorithm for regression analysis does not give a reasonable performance. Future changes to the algorithm may fix this drawback and consequently lead to an improved recognition performance.

5.4. Experiments and Results

The non-uniform frame spacing was tested with phonetic recognition experiments using the TIMIT database. Thirty nine features were extracted using the DCTC/DCSC feature extraction method (13 DCTCs and 3 DCSCs). Speech was sampled at 16 kHz. A frame size of 8ms with an initial frame space of 2ms was used to generate the spectrograms in all experiments unless otherwise stated. A Kaiser window with $\beta = 6$ was used as a window for each speech frame. As previously stated, the DCTC warping factor, α, was set at 0.15 for all experiments. For the HMM phase, a 3 state model with 16 mixtures was used.

The algorithms for frame deletion were used in experiments with varying threshold factors, C. The threshold factor C represents a percentage of the average rate of spectral variation. If the spectral variation at any point in the spectrogram is greater than the threshold factor, then the corresponding frames are kept. If not, the frames are discarded.

An additional condition of the number of frames allowed to be deleted in succession had to also be met. After the frame deletion algorithm was finished, a linear interpolation was performed on the spectrogram to return to the original number of frames. Interpolation was accomplished with the use of the "interp1" function in Matlab. As previously mentioned, the function was used in the default mode of 1-dimensional linear interpolation of the input data.

As mentioned in the DCSC section, the block jump is an integer number of frames over which the DCSC features are calculated. The frame space multiplied by the block jump yields the block space, or the time shift over which these DCSCs are calculated. A set of experiments were performed with a 1ms frame space and varying the block jump from 1 to 20. In other words, a block space from 1-20 ms was tested, which is an effective frame space from 1-20 ms, from the point of view of the HMM recognizer. It is important to note that the HMM was only presented a feature matrix, and was given the time spacing in ms between successive columns of the matrix, and considered this to be the frame spacing. The initial frame spacing used in the first step of spectral processing is essentially irrelevant to the HMM. Thus, in the majority of experiments reported in this thesis, the initial frame spacing was 2 ms and the block spacing was 4 frames, so the effective frame spacing (or block spacing) was 8 ms.

The purpose of the experiment reported here is to show that varying the frame spacing (from the viewpoint of the HMM), or block spacing, (from the viewpoint of our front end analysis program) does indeed have an effect on the recognition performance. It is important to note that the smallest possible frame space (more frames) is not necessarily the best. Consequently, selecting "good" frames becomes all the more important. Non-uniform frame spacing was not performed in these initial phonetic recognition experiments. The results are shown in the table below.

Table 12: Accuracy vs Block Spacing (ms)

Block space (ms)	Accuracy (%)
1	63.16
2	63.16
3	67.09
4	68.54
5	69.44
6	69.65
7	69.82
8	69.7
9	69.26
10	69.34
11	68.71
12	68.05
13	67.5
14	66.26
15	64.98
16	63.65
17	61.98
18	60.35
19	58.15
20	55.98

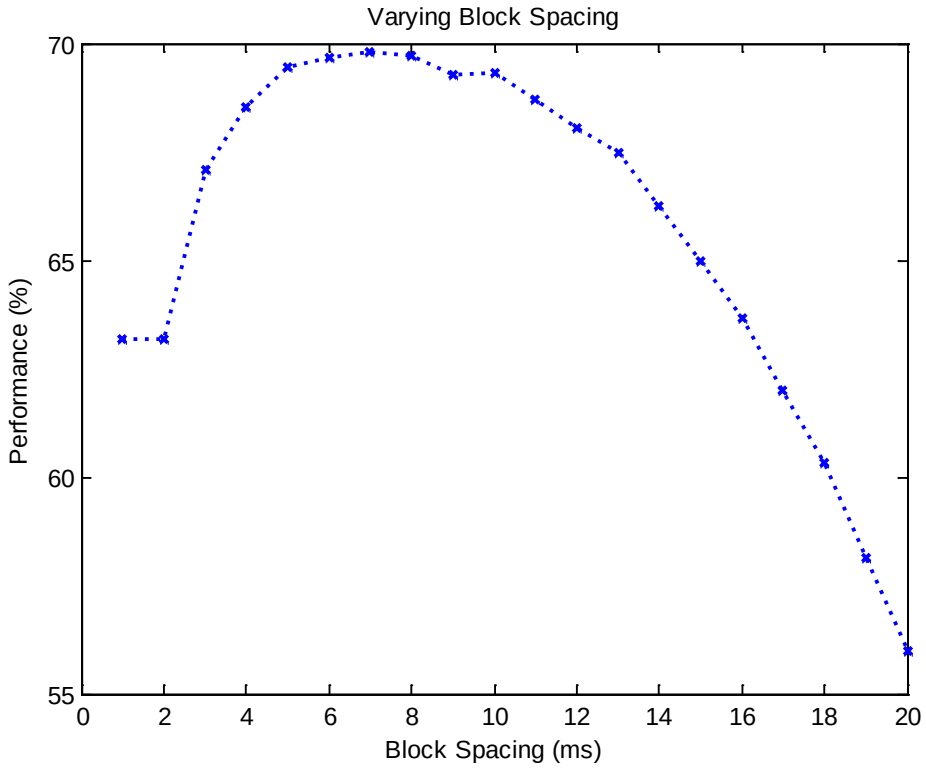

Figure 30: Plot of Accuracy (%) vs. Block Space (control—in ms)

5.4.1. L1-Norm Deletion Experiments and Results

The baseline accuracy for phonetic recognition was 68.17%. It should be noted that this baseline accuracy is not listed in the previous table because the log energy was not included in the generated features. The baseline performance was generated without the use of a deletion algorithm. A table comparing performance values of varying threshold factors, C, is given below for the L1-Norm deletion algorithm. Recall that C is the fraction of the overall rate of spectral variation of a particular utterance.

Table 13: L1-Norm Frame Deletion Accuracy vs Threshold Factor

Threshold Factor (C)	Accuracy (%)
0.05	68.28
0.075	68.51
0.1	68.4
0.15	68.27
0.2	68.22
0.3	66.15

From approximately 5% to 20%, an improvement over the baseline is observed, with the best performance at 7.5%. In other words, if frames that change faster than 7.5% of the average spectral variation are kept, then a small performance boost is observed.

An additional experiment using the L1-Norm frame deletion algorithm was performed on the feature matrix instead of the spectral values. In other words, instead of using the spectrogram as the focal point for measuring spectral variation, the DCTC feature matrix was used. An initial frame space of 4ms (block space 8 ms) was used. The results of this experiment are shown in the table below. (Baseline = 68.17%)

Table 14: L1-Norm Frame Deletion Accuracy of Feature Matrix vs Threshold Factor

Threshold Factor (C)	**Accuracy (%)**
0.1	68.24
0.125	68.24
0.15	68.27
0.2	68.30
0.25	68.39
0.3	68.16
0.4	67.87
0.5	67.93
0.6	67.71
0.7	67.6

The results suggest that reshaping the feature matrix did not benefit performance to any significant degree. It should be clarified that further testing should be done as this was not one of the original goals of this work, and thus only tested on a small scale.

5.4.2. Regression Analysis Experiments and Results

The results for the Regression Analysis experiments are given in this section. As shown in previous figures, varying the scaling factor, c, causes the spectrogram to change. A higher value for c yields a much more warped spectrogram relative to the original. The table below gives four different scaling factors (2.5, 5, 7.5 and 10) and the corresponding performances.

Table 15: Regression Analysis Accuracy vs Scaling Factor

Scaling Factor	Accuracy (%)
2.5	67.98
5	67.54
7.5	66.73
10	66.05

Recall the baseline of 68.17%. Although there is no boost in performance, it must be noted that greatly changing the spectrogram, in the form of a very high scaling factor, does not degrade the overall performance significantly. This means that a large amount of frames can be deleted without much degradation to the performance.

In the experiments for the Regression Analysis algorithm, as explained in section 5.3, the results were not as good as those obtained with the L1-Norm experiments. However, these experiments do add credence to the notion of non-uniform frame spacing. It shows that a proportion of frames can be non-uniformly spaced while still maintaining roughly the same effective performance.

Experiments with noisy speech (20dB SNR) were also conducted. The results demonstrate that for noisy speech, an even higher threshold factor can be used in the frame deletion algorithms compared to clean speech (80dB SNR). In other words, more frames can be deleted without a big degradation to the baseline performance of noisy speech. As previously expressed in section 2.6, non-uniform frame spacing has somewhat of a superior effect, relative to clean speech, when it comes to the amount of frames that are "allowed" to be deleted before there is any significant effect on the recognition performance. However, there were no significant improvements in the performance over the baseline. Further testing with a better measure of spectral derivative would be necessary to show recognition performance improvements.

5.5. Conclusions and Future Work

The results of this research demonstrate that there is potential for improvement in the area of non-uniform frame spacing in ASR. Although many methods have already been tested, there are still many more ways to approach non-uniform frame spacing. Particularly, the measure of spectral change is a fundamental topic that must be "solved." There are many ways to measure the spectral change of a speech signal. L1-Norm distance, delta coefficients and entropy are among these measures of spectral change. Additionally, the

criteria for frame selection in terms of a threshold value are also modular which opens up further possibilities for testing.

6. A TOOLBOX FOR A COMPLETE AUTOMATIC SPEECH RECOGNITION STSTEM

6.1. Overview

This toolbox constructs a complete ASR system. This toolbox is coupled with the forced alignment tool package, which has a separate manual. The forced alignment tool package is used to find the correct phonetic pronunciation transcriptions for a large dataset. To use the ASR toolbox, the basic assumption is that the correct phonetic transcriptions have been made available, either by running the forced alignment toolbox, or by manual labeling. The examples and steps in this tutorial assume that the phonetic transcriptions are created by the forced alignment toolbox. So, please run the forced alignment toolbox for the training database before implementing the examples in this tutorial. However, for other database where the transcriptions (in HTK MLF format) are already available, such as the Youtube database, it is also very easy to use this ASR toolbox without any need to do forced alignment beforehand. We will also briefly talk about this point in the training step. In this tutorial, our task is to build a system for Mandarin Chinese character level ASR using the toolbox. The steps involved in this task are:

1. Feature extraction for both training and test data.
2. Training monophones.
3. Training triphones.
4. Training a language model.
5. Decoding

The feature extraction step is exactly the same as in the forced alignment step, and the monophone training step has only a slight difference. Therefore, these two steps are only briefly explained. We focus mainly on steps 3 to 5. Additonally, near the end of this section of the report, a few experimental results are given.

Along with this manual, two folders are provided. In the folder “Tools,” there are all the tool matlab files and their setup files, and in the folder “files needed,” are all other files needed. These files are for all steps. As we go along steps 1-2-3-4-5, different files will be copied to our experiment folder.

Consistent with the tool format in the forced alignment package, each tool in this ASR toolbox is a matlab m file, and each m name begins with “Tool_...” A setup file is needed for each tool, and is the only argument that can be passed when the tool is called. The extension of all the setup files is “.dcf,” and for clarity, the setup file names also begin with “Tool_...” For example, the setup file for the tool “Tool_Decode.m” is “Tool_Decode.dcf,” and when this tool is called, the format is Tool_Decode(‘Tool_Decode.dcf’). The setup file for a tool contains all the control

options and parameters for that tool. The setup file and the matlab file should be placed in the same experiment folder. Next, steps 1 to 5 are illustrated.

6.2. Feature extraction (Tool_ComputeFeat.m)

6.2.1. Data preparation

As in the forced alignment step, before we run this feature extraction tool, some data preparation needs to be done. First, create a folder called “data.” Inside this folder, create a subfolder called “train_wave.” This is the folder where all the training wave files are to be placed. Copy all the training wave files into this folder. Similarly, create another folder called “test_wave,” and copy all the test wave files into this folder. Next, go outside “data” folder, and create a folder called “exp.” This will be the experiment folder. Then, copy "Tool_ComputeFeat.m" from “Tools” folder into “exp” folder. This matlab file is the feature extraction tool. Since we want to extract features for both training and test data, we need to call this tool twice, and each time pass a different setup file. Please copy “Tool_ComputeFeat_train.dcf” and “Tool_ComputeFeat_test.dcf” from “Tools” folder into “exp” folder. These are the setup files for extracting training and test features respectively. In addition, also copy "readhtk.m" file from "files needed" folder into "exp."

A list of all the wave files in “train_wave” and “test_wave” folders are needed for the feature extraction tool to read in wave files. Since it is assumed that the forced alignment step has already been run, which also requires the training wave file list in the feature extraction processing, we can simply create a folder “lists” inside the “exp” folder, and copy the training wave file list into “lists” folder. Then, rename this list to “train_wavefile.lst.” The same list used in the forced alignment step is called “wavefile.lst,” but here, we need to distinguish the training list from the test list. However, if we do not have this list from previous steps, we can simply create it using a short program provided in the “files needed” folder called “makelist.m.” Please copy this file into “exp” folder. The description of how to use this short program was explained in details in the forced alignment manual, feature extraction step. After running this program, the folder “lists”, and the wave file list will be generated. The folder name “lists” and the wave file list name “train_wavefile.lst” can be set in the program. Similarly, since we do not have the test wave file list yet, we can use “makelist.m” to create one and put it in “lists” folder also.

For simplicity, these two lists are also provided in “files needed” folder. We can simply copy them into “lists” folder if we do not want to manually create them.

6.2.2. Run tool

The detailed explanations of each option in the setup file is provided in the forced alignment manual. First, please refer to the feature extraction step in that manual. The only difference is that we need to call the tool (Tool_ComputeFeat.m) twice since we want to extract features for both training and test data. Assuming that you have already

read the procdure for the feature extraction step in the forced alignment manual, then, open a new matlab file in the experiment directory ("exp" folder), and write:

```
copyfile ('cp_MFCC.ini', 'tfront\cp_fea13.ini');
copyfile ('snr_801.trn', 'tfront\tfrontm.dat');
copyfile ('v7\tfrontm.exe', 'tfront\tfrontm.exe');
Tool_ComputeFeat('Tool_ComputeFeat_train.dcf');
Tool_ComputeFeat('Tool_ComputeFeat_test.dcf');
```

Then, save this new file as "do_main.m." This is similar to the main function in a C program, which is responsible for calling different tools in sequence. In our example, we choose to use the standard MFCC method provided in the tfrontm frontend. A brief description on how to setup tfrontm frontend is also given in the forced alignment manual, feature extraction step. Please read the corresponding steps first. The "do_main" function here assumes that you have already compiled the "tfrontm.m" file, thus, a "tfrontm.exe" was generated. In the forced alignment step, we used 42 features (39 MFCC features plus pitch), which was specified by "cp_42.ini," but in this example, we will use the standard 39 MFCC features without pitch first. So, the corresponding configuration file changed to "cp_MFCC.ini," which can be found in "files needed" folder.

Later on, we will also extract features with pitch (MFCC+pitch) in another experiment, and compare the performance without pitch features. In that experiment, we will use "cp_42.ini," which is also provided in the "files needed" folder. We need to choose which pitch tracker to use. The option "spare2" in "cp_42.ini" gives 3 methods: 1 for Yaapt, 2 for Yin, and 3 for Praat, and "spare1" controls all voiced mode or partially voiced mode in Yaapt. The option "Pitch" needs to be set to 1, in order to enable pitch tracking. If you want to use Praat pitch tracker, then, in addition to the steps illustrated in the forced alignment manual, tfrontm frontend steps, you also need to copy "praatcon" and "pitch.praat" files from "files needed" folder to "exp" folder. You can choose all voiced mode or voiced/unvoiced mode in "pitch.praat" file by removing or adding the # sign (which means to comment out the corresponding row).

After the feature extraction step, a "train_feat" and a "test_feat" folders will be generated inside "data" folder, and inside these two folders are the training and test feature files. In addition, a "train_wavefile.lst" and a "test_wavefile.lst" file will be generated inside "lists" folder. These are the lists of feature files for training and test data.

6.3. Training monophones (Tool_trainMono2.m)

This tool is for monophone training. Its setup file is "Tool_trainMono2.dcf." The training stage is almost the same as the same tool in the forced alignment step (Tool_trainFA.m). However, there are two major differences. One is the transcription preparation step; the other one is that there is an additional option called "Triphone_later" in the initialization step. We will mainly focus on these two differences in this section.

First, please copy the setup file "Tool_trainMono2.dcf" from "Tools" folder to "exp" folder. Next, create a subfolder "labs" in the "exp" folder. Then, go to the same "labs" folder in your forced alignment experiment folder, and copy the outcome of the last round forced alignment to "labs" folder of our current task example. If you have run the forced alignment step using the provided setup file (Tool_FA.dcf) in the forced alignment package, the last round outcome should be "aligned_6.mlf." This is the "perfect" version of the phonetic transcriptions of the training data. Rename this file to "trainphone_sp.mlf." This file is also available in the "files needed" folder, and you can just copy it from there to "labs" folder for convenience. But it is strongly recommended that you first run the forced alignment package by yourself to get familiar with similar tools.

A byproduct of the forced alignment step is a list of all the monophones in the training data, including the short pause "sp." We will use this list in our ASR task. Please go to "lists" folder in your forced alignment experiment folder, and copy the file "monophone_sp" to the "lists" folder of our ASR task. For convenience, this list can also be found in "files needed" folder.

Then, let's open "Tool_trainMono2.dcf" file and focus on the transcription preparation step. The purpose of this step is to delete the short pause (sp) between words from the transcription "trainphone_sp.mlf." The output is simply the transcription without "sp." The reason why we need both the transcription with and without "sp" is that the "sp" model is a short pause between every two words, so, this model is a one-state model, whose parameters are copied from and tied to the central state of the silence (sil) model, which is a 3-state model. To do this, at the beginning of training, a phone set of low mixture models (usually 1 mixture) will be trained using the transcription without "sp," and then, the "sp" model will be introduced by copying the central state of the silence model. After this point, we will use the transcription with "sp" to keep training the models. The forced alignment transcription contains the "sp" model. So, at the beginning of training, we need to delete it from the transcription.

The followings are the options to delete "sp".
Trans_prep: y means to turn on transcription preparation; n means to turn it off.
PhoneMLF_sp: this is the path of the transcription with "sp." It is the input of this step. In our setup file, the path is labs\trainphone_sp.mlf.
Conf_deleteSP: this is the path of the configuration file for the underlying HTK tool (HLEd) to conduct this deletion. In our example, please copy the file "deleteSP.led" from "files needed" folder to "toolconfs" folder. You should have already created the "toolconfs" folder inside "exp" folder in the feature extraction step if you choose to use HTK_MFCC or HTK_PLP frontend, as explained in details in the forced alignment manual, feature extraction section. "toolconfs" folder is where we put all the configuration files of the underlying HTK tools. However, since in our example, we've chosen to use the tfrontm MFCC frontend, which is not controlled by HTK, you may have not created this folder. So, please create "toolconfs" folder first if that's the case.

PhoneMLF_nosp: this is the path for the output transcription without “sp.” In our example, it is set to “labs\trainphone_nosp.mlf.” So, a MLF file “trainphone_nosp.mlf” will be generated in the folder “labs.”
PhoneList_nosp: this is the path for the output phone list without “sp.” In our example, it is set to “lists\monophone_nosp.” This list, along with the transcription, will be used to train the phone set without “sp.”

Notice that the operations in the transcription preparation step are based on the assumption that the phonetic transcription with “sp” is generated by the forced alignment package. However, in some cases, the transcription is already available in MLF format, such as the Youtube database, and it is possible that there is even no “sp” in the transcription. In this case, there is no need to delete any “sp.” However, we still need a list of all the phones encountered in the transcription. So, we can follow the following steps:
a. Set “Trans_prep” to y, since we still need to generate the phone list.
b. Set “PhoneMLF_sp” to the path of your phonetic MLF file. Note that do not try to change the parameter name “PhoneMLF_sp,” because otherwise, the code of the tool will also need to be changed. Though the parameter name is “PhoneMLF_sp,” that is only for clarity. You need to be aware that in this case, this is just your phonetic transcription MLF path.
c. For “Conf_deleteSP,” you can still leave the configuration file path “toolconfs\deleteSP.led” here. Since there is actually no “sp” in the transcription, the underlying HTK tool (HLEd) won’t do anything though the command in the file “deleteSP.led” (DE sp) means to delete “sp” from the transcription. The other way is to make the file “deleteSP.led” an empty file.
d. For the output file parameter “PhoneMLF_nosp,” this is exactly the same as the input file parameter “PhoneMLF_sp” because there is no “sp” involved. Suppose your input MLF file name is “trainphone.mlf,” you can make the output file name “trainphone1.mlf,” but they are exactly the same files.
e. “PhoneList_nosp,” in this case, is the list of monophones in the transcription, and is what we really want. You can specify the file path as you want here.

This is how the transcription preparation step generalizes to the case where the MLF is not obtained from the forced alignment. In the training stage, there are a couple of other places that require inputs of the MLF transcription or phone list, both with and without “sp.” So, we can simply provide the same file for the “sp” version and “non-sp” version. For example, in the embedded training stage, we can set the entries “hmmList_nosp” and “hmmList_sp” both to the path of our sole HMM list. In addition, remember to set the entry “fix_sil” to n, because we do not introduce the “sp” model in this case. The meaning of “fix_sil”, as well as how the “sp” model is created, and how the silence model is fixed, are described in details in the forced alignment manual. Please refer to “Tool_trainFA” section for details.

The transcription preparation in this training tool assumes that a MLF format transcription is already available, such as the “trainphone_sp.mlf” file in our example, and any editing should be made based on this MLF file. However, it is often the case that

the initial transcription is in its “raw” format, not MLF format. The “raw” format may take on many possible patterns. For example, in TIMIT database, there is one phonetic transcription for each wave file, and the transcription has time markers for each phone. The reason why we start from a MLF is that it is very difficult to use the same piece of code to convert different raw formats into a MLF format. The code to make this conversion may vary, depending on the specific raw format, and it also makes the tool setup file very cumbersome with so many options for different variations. So, we have to assume that a user has already got this MLF format of transcription. It is recommended that you go through the tool “HLEd” in HTKbook, section 17.10 to learn how to make this MLF file. An example for TIMIT database is as follows:

In the matlab command window, type in:
arg = sprintf('-G TIMIT -i %s -l * -n %s -S %s %s', 'trainphone.mlf', 'hmmList','trnp_exsa.lst','phn2cmu39.led');
system(sprintf('HLEd -A -T 1 %s', arg));

Then, a MLF file “trainphone.mlf,” and a HMM list “hmmList” will be generated in your current matlab directory. One input is a list “trnp_exsa.lst,” which is a list of all the raw phonetic transcriptions with one transcription for one wave file. Each file path in this list is with respect to your current matlab directory. Another input is a configuration file “phn2cmu39.led,” which is provided in the “files needed” folder. It converts the TIMIT original phone set to the CMU phone set. A “phn2lab48.led” configuration file is also provided, which converts the TIMIT original phone set to the 48 phone set. A user can choose which phone set to be trained. These input files should also be placed in your current matlab directory. “-G TIMIT” means that the time markers in the raw transcription files are in TIMIT format, whose unit is sample index. After conversion, the base unit will change to HTK format, which is 100ns.

With this MLF file as well as the HMM list, there is no need to make any further editing in the transcription preparation step of the monophone training tool, since the original phone set has already been converted to the 39 phone set by the commands above. So, you can simply set “Trans_prep” to n. Whenever the tool requires the MLF or HMM list in the setup file, either with “sp” or without “sp,” simply pass the same MLF and the HMM list path to both places.

This ASR toolbox is designed for a large database. The training method for a large database includes two steps: flat start initialization and embedded training. The concepts of flat start and embedded training are explained and illustrated in the forced alignment manual. Both steps do not require time stamps in the phonetic transcriptions. For some small databases, such as TIMIT, the phonetic transcriptions have time stamps for each phone. For such a database, the training method also consists of two steps: boost initialization and embedded training. The boost initialization uses the time markers to cut out feature segments for each phone, and each phone model is trained by its own feature segments individually. The underlying HTK tools for the boost initialization are HInit and HRest. An overview of the boost initialization algorithm can be found in HTKbook, section 2.3.2. At this point, this ASR toolbox only supports the flat start+embedded

training mode, and this is also why the monophone training tool is called "Tool_trainMono2." A "Tool_trainMono1" will be developed, which only supports the boost+embedded training mode, which is the typical training method for small databases whose phonetic transcriptions contain time markers.

A new option "Triphone_later" is added to the initialization step, compared with the training tool in the forced alignment package. If we want to use monophones to form triphones in the next step, we should set "Triphone_later" to y. Triphones can only be formed by 1-mixture monophones. In other words, we can not first train a set of 16-mixture monophones, and use them to form a set of 16-mixture triphones. We can only start from 1-mixture triphones, which are built from 1-mixture monophones, and then, the mixture splitting will be conducted to these 1-mixture triphones to split the mixtures till the desired order is reached. The triphone mixture splitting sequence is specified in the triphone training tool (Tool_trainTri.m). So, if we set "Triphone_later" to y, the monophone mixture splitting sequence will not have any effect. The number of mixtures given by "numMixture" will be forced to 1, though it is specified as 1;2;4;6;8;12;16 in the setup file. Also, only the first number of the iteration sequence will be preserved, though it is set as 3;5;5;6;6;7;7. So, all the monophones will be 1-mixture phones, and 3 iterations will be conducted to train these monophones. Thus, in a word, setting "Triphone_later" to y in the initialization step guarantees that all monophone models are 1-mixture models. Later on, these 1-mixture models will form triphones. If triphones are not wanted, we can simply set "Triphone_later" to n. In this case, the mixture splitting sequence will be performed, and multiple mixture monophones will be generated.

All the other options and parameters in the setup file are the same as those in the monophone training tool of the forced alignment package. Please read through and implement section 2, Tool_trainFA in the forced alignment manual. In our task example, a set of 1-mixuture monophones will be generated in the folder "hmms\fhmm_mono."

6.4. Training triphones (Tool_trainTri.m)

After we've got a set of 1-mixture monophones, triphones will be made from these monophones. There are two types of triphones: internal word triphones and cross word triphones. Before we get into the details of the tool, let's first get to know what they are.
The formation of internal word triphones is bounded by the inter-word short pause "sp" and the silence (sil). Here is an example of how to convert monophone transcription into internal word triphone transcription:
Original sentence: sil this sp man sp…
Monophone sequence: sil th ih s sp m ae n sp…
Internal word triphone sequence: sil th+ih th-ih+s ih-s sp m+ae m-ae+n ae-n sp…

In the above example, the monophone sequence is converted to the internal word triphone sequence. A triphone consists of a central phone, a left phone, and a right phone. For example, in the triphone "th-ih+s", "ih" is the central phone, "th" the left phone, and "s" the right phone. In HTK, we use a "-"to represent the left context, and a "+" to represent the right context. Note that the word boundary markers "sil" and "sp" are not used to

form triphones. They block the addition of context at word boundaries, such that some biphones (or monophones) will also be generated. For this reason, when we talk about internal word triphones from now on, we will simply call them triphones, but reader should be aware that the concept of "triphones", in the internal word case, also includes biphones or monophones.

The formation of cross word triphones is not subject to the word boundaries. This is why it is called cross word triphones. Next, let's look at the same example:

Original sentence: sil this sp man sp…
Monophone sequence: sil th ih s sp m ae n sp…
Cross word triphone sequence: sil sil-th+ih th-ih+s ih-s+m sp s-m+ae m-ae+n ae-n+…

As can be seen from this example, the formation of cross word triphones is not restricted to "sil" and "sp." "sil" is regarded as the context of the center phone, whereas in the internal word style, "sil" can not be a part of any triphones. When building triphones, the "sp" is "jumped over", so that monophones from adjacent words are combined into a triphone, such as "ih-s+m." However, the "sp" is not totally ignored. It is actually shifted to the right by one phone. There are different ways to deal with "sil" and "sp" when making cross word triphones. The method in the above example is adopted by the RMHTK, which is an example of how to build HTK based systems for the ARPA RM task. It can be downloaded from http://htk.eng.cam.ac.uk/download.shtml. Another way of making cross word triphones can be found from www.keithv.com/software/htk/. In our example, we will use the method in the above example.

After knowing what triphones are, we will get into the procedures of triphone training. We will start from internal word triphones.

6.4.1. Internal word triphones

Copy "Tool_trainTri.m" from "Tools" folder to "exp" folder, and copy the setup file "Tool_trainTri_inword.dcf" from "Tools" folder to "exp" folder. You may notice that there is another setup file "Tool_trainTri_xwd.dcf." That is the setup file for cross word triphones. Training internal word and cross word triphones follows the same steps. So, only one tool (Tool_trainTri.m) covers both cases. Different setup files will be passed to the tool respectively.

Next, please open "Tool_trainTri_inword.dcf." We will explain tutorially the theory behind each step as we go through the setup file.
Trace_on, Clean_up, LogDir: these three terms are the same as in the monophone training. "Trace_on" enables the progress to be displayed on the screen. "Clean_up" deletes the old triphone models in each step before new models are generated. "LogDir" specifies a log directory inside which a progress report "progress_trainTri.log" will be generated.

To train internal word triphones, we must first have internal word triphone transcriptions. In the first example above, we have learnt how to convert monophone transcriptions to the internal word triphone version in theory. This conversion is implemented by the transcription preparation step. This step has two inputs and two outputs.

Trans_prep: set this to y enables the conversion, and n turns it off.
PhoneMLF: this is the path of the monophone transcription MLF file, which is an input. Note that in the setup file, this is set to "labs\trainphone_sp.mlf." This is the transcription with "sp" between each two words. To make internal word triphones, there must be a word boundary marker, such as "sp."
Conf_mon2tri: this is the path of the configuration file for the underlying HTK tool (HLEd) to convert monophones to internal word triphones. Please copy the file "mktri_inword.led" from "files needed" folder to "toolconfs" folder. In "mktri_inword.led" file, there are two commands: each WB specifies a word boundary marker, and TC means to expand the monophone transcription to triphone transcription.
TriMLF: this is the output triphone MLF path. We set it to be "labs\traintri_inword.mlf."
Trilist_ini: this is another output, which is a list of all the triphones in the transcription TriMLF. Note that this list only covers the triphones in the training data. But many triphones, which are not in this list, may appear in the test data. To solve this problem, we will have another full list in a later step. So, at this point, this list will only be used for triphone initialization, as implied by the name of the term "Trilist_ini."

After we obtain the internal word triphone transcription, we can begin the training processing. There are two main steps: initialization and making tied state triphones. We will describe both steps later. But before we get into the details, there are some global settings and files needed by both steps.

Train_on: y turns on training; n turns off training.
Feat_List: this is the training feature file list. Both steps will need this list.
Tri_MLF: this is the triphone transcription MLF file generated in the transcription preparation step. Both steps will need this transcription.
Src_hmmfolder: this is the monophone HMM folder, which is the starting point of making triphones. Our monophones are stored in "hmms\fhmm_mono."
Final_hmmfolder: this is the folder to store the final triphone models after initialization and making tied state triphones. This folder is the end point of training. We will place the final triphone models in "hmms\fhmmtri_inword." This directory will be automatically created.
embdOptStr: this is the pruning threshold sequence of the embedded training, which has the same meaning as in monophone training.
Conf_embd: this is the configuration file path for the underlying HTK tool (HERest) of embedded training. There should be a "herest.conf" file already copied to "toolconfs" folder in the monophone training step.

Next, we come to the initialization step. The initialization of triphones is somewhat analogous to the flat start initialization of monophones. First, HTK will read in a list of triphones to be initialized. Then, for all the triphones in this list that have the same central

phone, HTK will make a copy of the parameters of the central phone (which is a monophone), and use it as the initial parameters of all these triphones. For example, for all the triphones of the format *-b+* (this includes *-b+*, *-b, b+*, b), the parameters of monophone b will be used as the initialization of all triphones of *-b+*. Then, embedded training will be conducted for a couple of iterations, and these initial parameters will change correspondingly.

A problem is obvious: the number of triphones is huge. If each triphone requires its own samples to be trained, then, the parameter estimates will be very poor since many triphones only appear once or twice. So, in the initialization processing, in addition to the "clone" operation described above, the transition matrices of all the triphones in the class *-b+* will be tied together. Generally speaking, tying means that one or more HMMs share the same set of parameters. When reestimating tied parameters, the data which would have been used for each of the original untied parameters is pooled so that a much more reliable estimate can be obtained.

Of course, tying could affect performance if performed indiscriminately. Hence, it is important to only tie parameters which have little effect on discrimination. This is the case in the initialization where the transition parameters do not vary significantly with acoustic context but nevertheless need to be estimated accurately. Some triphones will occur only once or twice and so very poor estimates would be obtained if tying was not done.

With this background, let's go into the setup file to see the initialization part.

Init: y turns on the initialization; n turns it off.
Iteration_init: this is how many iteration of embedded training to be conducted after initialization.

There are three inputs for initialization.

hmmlist_mono: this is the monophone HMM list path. Since the initial parameters of each triphone is copied from its central phone, we need to pass this monophone list to HTK to specify which monophones to copy from. Note that in the setup file, this list includes the "sp" model, because "sp" is also a member in the triphone list, and so it needs to be carried through all stages.
Trilist_init: this is the triphone HMM list path. This list is generated in the transcription preparation stage. The triphones in this list are to be initialized.
Conf_init: this is the configuration file path for the underlying HTK tool (HHEd) to copy parameters as well as tie transition matrices. Please copy the file "tieTrans.hed" from "files needed" folder to "toolconfs" folder. Let's look at a command in this configuration file: TI T_zh {(*-zh+*,zh+*,*-zh).transP}. The command "TI" means to tie a parameter; {(*-zh+*,zh+*,*-zh).transP} means that the parameter to be tied is the transition matrices (specified by transP) of all the triphones whose central phone is zh. The "T_zh" is the macro name of the tied matrix, which means for all the triphones in this class, they will have an identical transition matrix called "T_zh." Note that there is no "sil" and "sp" in

these commands, since “sil” and “sp” have no left or right contexts according to the internal word rule of forming triphones. But it does not matter if they are in this file; for example, there is a command TI T_sp {(*-sp+*, sp+*,*-sp).transP}. In this case, a warning will show up, saying there is nothing to tie for this class, but it won’t hang up the program.

It is very easy to write a short program to create this configuration file by passing it a monophone list (without sp and sil), and each monophone will be expanded into a command of the form “TI…” The content of this configuration file changes with the monophone list. So, if you have your own phone set in your own task, please write a program to make this configuration file by yourself.

However, at this point, the configuration file only does the tying job by executing each “TI” command. It does not include the cloning job, as described above. Actually, what the tool does is that it will add a row before all the “TI” commands, which is “CL lists/trilist_ini_inword,” and rename this complete configuration file as “mktri.hed,” and save it to our experiment folder “exp.” The command “CL” simply means clone. The initial triphone list “lists\trilist_ini_inword” is specified by the “Trilist_init” entry in the setup file. Then, the tool will use “mktri.hed” as the complete configuration file to conduct both cloning and tying.

There are two output entries in the initialization step:

TgtDir_init: this is the directory to save the initialized triphones. A folder “hmms\hmm3_init_inword” will be automatically created as specified in the setup file.
Stat_embd: this is a statistical file generated by the embedded training. As indicated in the setup file, 3 iterations of embedded training will be performed after initialization. After each iteration, a statistical file “toolconfs\hstats” will be generated. The newer one will overwrite the old one. The last round statistical file will be used in the next step, which is making tied state triphones.

Next, before we go into the tool implementation of the last step, a theoretical overview is very helpful for us to understand what this step does.

The outcome of the initialization step is a set of triphones with all triphones with the same central phone sharing the same transition matrix. However, there are two problems remaining. First, the data insufficiency problem is still prominent, since only the transition matrix in each triphone class is tied. When estimating these models, many of the variances in the output distributions will be floored. Second, the triphones that have been trained so far are all from training data; but many possible triphones that might appear in the test data are missing from the training data. To solve these two problems, the last step in building triphones is to tie states within each triphone set (class) that has the same central phone, so that data will be shared when estimating the parameters of each tied state.

Unlike the initialization step, which ties the transition matrices of all the triphones in the same class, how to partition the states of all the triphones with the same central phone into different clusters requires phonetic knowledge. In simple words, states of triphones with similar co-articulation effect is most likely to be tied together. A common characteristic of the triphones in the same cluster is that the co-articulatory impact of their left and right contexts to the central phone is similar across the cluster.

A decision tree based method is used to decide which triphones are to be put in a cluster. First, a question set is carefully designed according to similar co-articulatory impact of the left context and right context. Each question is a binary question. Let's give two examples of English question set. Here is one example of a left context question:

QS "L_Class-Stop" {p-*,b-*,t-*,d-*,k-*,g-*}
"QS" means that this is a question, and "L_Class-Stop" is the question name. This question simply asks: is the left phone of the central phone a stop consonant, namely, one of p, b, t, d, k, g? Recall that HTK uses "-" to represent the left phone of a central phone. Here is another example of a right context question:
QS "R_Nasal" {*+m,*+n,*+ng}
This question asks: is the right phone of the central phone a nasal consonant, or namely, one of m, n, ng? It can be seen that each question represents a group of phones that have similar impact on the pronunciation of the central phone.

The decision tree is constructed based on these questions. For example, suppose we want to make clusters of state 2 of all the triphones with a central phone "aw." Before clustering, there might be hundreds of triphones in this set. Initially, state 2's of all the triphones in this set are pooled at the root of a tree (the tree has not been built yet). Then, the question set is loaded. Since the answer to each question is binary, by answering a question, the initial pool will be split into two pools. Splitting any pool into two will increase the log likelihood of the training data for that state, since it provides twice as many parameters as the original pool (because each sub-pool has its own Gaussian output density) to model the same amount of data. After all the questions have been scanned through, the one question that provides the biggest improvement of the log likelihood of the training data will be selected as the first branch of the decision tree. This branch gives the best split of the root node, and two descendent nodes are generated by this splitting. The first part of the original pool is placed at one descendent node, and the second part is at the other descendent node. Then, the pool at each descendent node is split again by the locally optimized question (locally means the question maximizes the improvement of the likelihood of training data at that local node). Thus, this processing repeats and a decision tree is built by this top-down sequential optimization process. As the tree keeps growing, the maximal improvement of the log likelihood of the training data brought by a splitting at a descendent node gets smaller. When the maximal improvement at any node falls below a user defined threshold, the construction of the tree stops. It is easy to see that a smaller threshold makes the splitting process last longer, and thus, makes the tree bigger. When the process stops, the nodes at the very bottom of the tree, who do not have any descendants, are called leaf nodes, or senones. The states at each senone are tied together. From now on, these states will share the same training data, and have identical

output density parameters. A more detailed explanation of the decision tree based method which includes a pictorial example can be found in the HTKbook, section 10.5. A basic paper on this tree-based tying can be found in [77].

There are a couple of points revealed from the descriptions above. First, the design of the question set is crucial for the quality of the tying. Ideally, the question set would include every possible context which can influence the acoustic realization of the central phone, and can include any linguistic or phonetic classification which may be relevant. There is no harm in creating unnecessary or "meaningless" questions, since only the questions that give the maximal increase in the log likelihood of the training data will be selected in the tree construction processing. Second, the user defined stop criteria determines the number of tied states (leaf nodes). The smaller it is, the more splitting will be conducted, and therefore, more tied states will be generated. In a triphone system, the number of triphones is not that important. What matters is the number of tied states, because they are what get trained. So, this user defined threshold needs to be tuned according to the amount of data we have.

Finally, another advantage of this decision tree based clustering is that it's able to synthesize the triphones which never appear in the training data. In other words, even if a triphone does not have any samples in the training data, it can still be trained. This is because after the state tying, there is actually no longer a concept tied to any individual triphone. The states of each triphone are categorized into different clusters. So, for any triphone which is not in the training data, each of its states will first find the corresponding tree, and then descends that tree by answering the questions at each node until it gets to one of the leaf node. Then, it will use the parameters of this leaf node as its own estimates for that state.

With this background, let's focus on the tool to implement these algorithms. Please focus on the "Tied state triphones" part of the setup file.

Tie: y turns on the state tying; n turns it off. Once it is turned on, the models will be loaded from "TgtDir_init," which is the target folder of the initialization step, and the tied models will be saved in "Final_hmmfolder," which is the folder to store final HMM models.
Iteration_tie: this is how many iterations of embedded training will be performed after making tied state triphones.

Let's skip the two configuration files specified by "Question" and "TB" for the time being, and focus on the "Full_list" entry as one of the inputs. As stated in the background, after the decision trees have been built, unseen triphones in the training data will be synthesized using the trees. To do this, we need to manually create a "full list" which contains all possible internal word triphones for our ASR system, so that HTK will collect those unseen triphones with respect to the training data from this full list. Since the vocabulary of an ASR system consists of all possible words that can be decoded in theory, we can easily find all possible internal word triphones by converting the

monophone pronunciation of each word in the dictionary to its internal word triphone form, and make a list of it. To make this conversion, please follow the following steps:

a. Create a folder called “dicts” inside “exp” folder, and copy the dictionary file “dict863_tone_sp1” from “files needed” folder to “dicts” folder. Note that a “sp” is appended after each pronunciation entry.
b. Copy the configuration file “global.ded” from “files needed” folder to “exp” folder.
c. Make sure that your matlab directory is “exp.” Then, in your command window, type in this command:
system(sprintf(‘HDMan –b sp –g %s %s %s’, ‘global.ded’,’tridict’,’dicts\dict863_tone_sp1’));

The command “HDMan” is the HTK tool to manipulate a dictionary, and the option “-b sp” specifies “sp” as the word boundary. Note that the original dictionary to be converted must be in sorted order; otherwise, “HDMan” won’t work. The dictionary provided is already in sorted order. Then, a file “tridict” will be generated inside “exp” folder. This is the internal word triphone dictionary. The monophone pronunciations have been converted to its internal word triphone format. You may find that the Chinese words all become numbers. Those numbers are the encoding of the Chinese characters. But this does not matter because what we want is the triphone pronunciations, not the Chinese words. Then, you can write a program to get rid of those Chinese words and only preserve the triphones. Such a matlab program called “remove.m” is provided in “files needed” folder. Copy this file to “exp” folder, and the words will be eliminated after running this program. Then, use a text editor to open the output file “tlist,” and replace all spaces by line breakers (\n) and remove empty lines. However, this list contains duplicated entries. Please copy the file “tlist” to a Linux system, and open a terminal. Then, type in the following two commands:

```
dos2unix tlist
awk ‘!a[$0]++’ tlist>fulllist_inword
```

Then, a “fulllist_inword” will be generated. Only one entry of each duplicated entries is preserved. Copy this file back to the “lists” folder inside the “exp” folder. Note that in HTKbook section 3.3, the tutorial example, there is a “-n fulllist” term in HDMan command. A “fulllist” will be generated after the conversion of the dictionary. However, please do not use this method, since this “fulllist” misses some triphones, which will result in errors in following steps. For your convenience, a “fulllist_inword” is provided in the “files needed” folder using the recommended method above. This full list comes from the dictionary “dict863_tone_sp1.” In your own project, you need to make your own full list if a different dictionary is used.

Full_list: make a full list of all possible internal word triphones from the dictionary, and specify its file path here.

Now, let’s go back to the two configuration files Question and TB.

Question: this is the question set for building the decision tree. An example question set formed by Chinese tonal phones is provided. Please copy the file "Quest.hed" from "files needed" folder to "toolconfs" folder, and specify the file path here. There is an English question set. It can be found in HTK samples\RMHTK\lib\quests.hed. The HTK samples can be downloaded from the official website of HTK.
TB: please copy "TB.hed" file from "files needed" folder to "exp" folder. Let's look at one of the TB commands in this file:
TB 2000.0 "zh_s2" {("zh","*-zh+*","zh+*","*-zh").state[2]}
Each TB command constructs a decision tree for a state of a triphone set. In this command, a decision tree will be built for state 2 of the triphone set with the central phone "zh." The procedures of building this tree is described in the background section. The number 2000 is the user defined stop criteria for the growth of the tree. This number needs to be tuned according to the amount of training data. In our example, 2000 is the optimal setting for tonal internal word triphones.

A Perl script "mkclscript" can be found in HTK samples\RMHTK\perl_scripts to generate this TB file. It is run under Linux. The threshold number as well as a monophone list excluding "sil" and "sp" are needed as two arguments of the program.

In addition to the full list of all the possible triphones, there are other two inputs:

Stat_embd: this is the statistical file generated by the last round of the embedded training after the initialization step. This file needs to be loaded at the beginning of the clustering processing. After the state tying, it becomes useless, and will be overwritten by embedded training.
Trilist_init: this is the initial triphone list file path. The clustering will be conducted to each triphone in this list.

In addition to a new set of tied state (also tied transition matrices) triphones stored in the "Final_hmmfolder," another two outputs will also be produced:

Trilist_tied & Tree: "Trilist_tied" is the output file path for a list of all compact tied-state triphones. In our example, it is set to "lists\tiedlist_inword." "Tree" the path where the decision trees get saved to. To be specific, before making decision trees, the tool will first merge the question set "Quest.hed" and the TB file "TB.hed" together, and a new file "tree.hed" will be generated inside "exp" folder. Then, it writes in the following 3 lines at the end of "tree.hed." Then, "tree.hed" will be used as the complete configuration file for the decision tree construction as well as state clustering.
AU "lists/fulllist_inword"
CO "lists/tiedlist_inword"
ST "toolconfs/trees_inword"

"AU" means to synthesize all the unseen triphones in the full list after trees have been built. After state tying, it is possible that for some triphones with the same central phone, their three emitting states all fall into the same leaf node in corresponding trees. Thus, these triphones become exactly identical (recall that their transition matrices are already

tied together, thus also the same). The "CO" command finds such triphones and tie them together, producing a new list of models whose path is designated by "Trilist_tied." This final list will be used to load HMMs in the following embedded training and decoding steps. Finally, the generated decision trees will be saved to the path "toolconfs\trees_inword."

Till now, we have completed making 1-mixture triphones. These 1-mixture triphones are saved in the directory "hmms\fhmmtri_inword" specified by "Final_hmmfolder" entry. However, if we want to get multiple mixture triphones, we can carry out the mixture splitting processing, controlled by the last 4 entries in the setup file. The splitting is directly conducted to the models in the folder "Final_hmmfolder."

Split: y enables mixture splitting; n disables it.
numMixture & Iteration: these are the mixture splitting sequence as well as the number of iteration sequence.
numState: this is the number of emitting states.

At this point, the models in the "hmms\fhmmtri_inword" folder become 16-mixture models, as specified by the mixture splitting sequence. We need to emphasize again that multiple mixture triphones can ONLY be obtained by splitting mixtures from 1-mixture triphones. Directly making multiple mixture triphoens from multiple mixture monophones is not permitted. In fact, when the tool loads monophones before triphone initialization step, it will first check whether the monophones are 1-mixture. If not, the tool will give out an error and stop proceeding.

6.4.2. Cross word triphones

The theory of making cross word triphones is the same as that of internal word triphones. There are two differences. First, the conversion from monophones to cross word triphones is not subject to the word boundary marker "sil" and "sp." Second, due to the first property, the full list in making tied state triphones can not be obtained from the dictionary. We will only focus on these two differences. All other steps are the same as in the internal word triphone case.

Please copy the setup file "Tool_trainTri_xwd.dcf" from "Tools" folder to "exp" folder. This setup has exactly the same format and entries as the setup file for the internal word triphones. So, it can be called by the tool in the same way:
Tool_trainTri('Tool_trainTri_xwd.dcf');

The first different place is in the transcription preparation step:

Conf_mon2tri: this is the configuration file path for the conversion between monophones to cross word triphones. Please copy the file "mktri_xwd.led" from "files needed" to "toolconfs" folder, and specify the file path here. "mktri_xwd.led" makes cross word triphones from monophones in the same way as in the example above.

The second different place is in the tied state triphone step:

Full_list: since the cross word triphones can be formed by monophones across adjacent words. The full list of the system cannot be extracted from the dictionary. We need to find all possible cross word triphones in the task language in a brute force way, and put them in the list. Making this list requires linguistic knowledge. Given a list of all monophones, randomly combining three of them into a triphone is not a good way, because certain combinations are illegal in a language. These illegal triphones will become burden of the system since they require more training data, and will never be used in a recognition network. In a Chinese system, all possible triphones can be found by the rules of how Initials and Finals form a syllable.

Specifically, the first rule is that a syllable in Chinese is formed by an Initial and a Final. The Final always goes after the Initial. The pronunciation of a character is specified by a syllable, and a word is formed by characters. So, when creating triphones, if the middle phone is an Initial, then, its right context can only be a Final, and its left context can be either "sil" (if this character is the first character of a sentence) or a Final. Similarly, if the middle phone is a Final, then, its right context can be any Initial or "sil" (if this character is the last character of a sentence), and its left context can only be an Initial. Another rule is that there are only certain Finals which are allowed to append an Initial when forming a syllable. For example, for the Initial "j," the Final "ong" is invalid to form a syllable, but "iong" is a valid Final for "j." So, knowing the valid Final sets for each Initial, we can further rule out a lot of invalid triphones.

A full list called "fulllist_xwd" made by these rules is provided in "files needed" folder. Please copy this file to "lists" folder. This list contains all valid triphones in Chinese made from the monophones covered by our task. There are one or two monophones not covered by our task database, but are valid Chinese monophones. So, this full list is not really 100% complete, but is appropriate for our task.

For a language which does not have too many monophones, such as English, it is fine (though not recommended) to "brute force" use all possible combinations, regardless of any linguistic rules. A Perl script can be found in HTK samples\RMHTK\perl_scripts\full_list.prl, which generates all possible monophones, biphones, and triphones for a cross word system from a monophone list (the monophone list should remove sil and sp first). Minor change to the program can be made to output only those cross word triphones.

All the other entries in the cross word triphone setup file are the same as the internal word one, except some file or folder names are appended by "_xwd" to distinguish from the internal word ones. So far, all the acoustic aspects of our ASR system have been completed.

6.5. Language modelling (Tool_trainLM.m)

In the previous steps, we have generated a set of acoustic models. In this step, we will focus on language modelling. The tool for language modelling is "Tool_trainLM.m," and its setup file is "Tool_trainLM.dcf." Please copy these two files from "Tools" folder to "exp" folder.

Though, in this manual, language modelling is placed in the fourth step, it's recommended that you run this tool apart from other steps. Do not connect this step with others. First, the language model is not likely to change with other steps. We could have different features, or different acoustic models, but these changes won't lead to any changes for the language model. So, it is usually the case in a large vocabulary ASR system that a language model should be prepared before other steps. Once it has been created, it does not change with other components. Second, a language model is a complete system by itself. It can be trained and evaluated isolated from any acoustic factors. It has its own training data and test data, both in text format. In other words, we do not need a complete ASR system to either train or tune a language model. So, people always train and tune up a language model towards different tasks without any acoustic data, and then put it in the whole system. Third, all language models for an ASR system comply with a set of standard format (ARPA format). There are various software specialized on language model analysis, such as SRILM and CMU_Cam_Toolkit. They provide much more powerful and flexible choices for making a language model than HTK. So, people always use such language-model-specialized software to generate better language models, and then connect them with other components of the ASR system in HTK environment. This connection is enabled by the standard format of language models.

No matter which software is selected, a background of language modelling is important. It is too long to give a detailed background in a manual for a complete system. Please refer to Chapter 14 of HTKbook for fundamental knowledge on language modelling. We will get into the task description and tool description directly.

In our example, the training data of the language model is a subset of the transcriptions of the acoustic wave files. The sentences in the wave files are divided into 4 groups (A,B,C,D). We use all speakers who speak A, B, C, and some speakers who speak D as the training data for the acoustic model, and use the rest of speakers from D as the test data. The language model training data is the transcriptions of A, B and C. But we also add the words in D into the vocabulary of our language model. There are many words in D that are unseen in A, B, C. These unseen words will still be assigned unigram probabilities according to the smoothing algorithm, which means in theory, it is at least possible for them to be correctly decoded.

This tool supports two methods. The first one uses the transcription of the acoustic data to build a simple bigram model. The second method uses plain text from any sources to create any n-grams specified by user. Let's first look at method 1.

Please open “Tool_trainLM.dcf.” Before getting into method 1, there are 4 global settings for the tool:

Trace_on: y enables envision of progress on the screen; n turns it off.
Train_on: y turns on LM training; n turns it off.
Convert_on: this step is to convert an ARPA format LM to a lattice format. In the decoding step, the decoder Hvite works with the lattice format of a language model, not its ARPA format. For a detailed description what a lattice network is, please refer to HTKbook section 12.2. The other decoder HDecode works directly with the ARPA format. So, there is no need to make this conversion is HDecode is to be used in the next step.
Log: as in other tools, a progress report “progress_trainLM.log” will be generated inside the folder specified here.

6.5.1. Method 1

Method 1 is a relatively simple method to create a LM. The source data usually comes from the transcriptions of the wave files, one transcription in one separate file, with one word in one row. The tool will first convert these “raw” transcriptions into a MLF file, and then, the LM is built from the MLF file.

One major restriction for this method is that we cannot pass it a predefined vocabulary. By default, this method uses all the words encountered in the transcriptions. The vocabulary is the first important step in building a LM. In theory, for those out-of-vocabulary (OOV) words in the source data, they should be either mapped to an unknown class, or simply thrown away. For those words in the vocabulary, but not in the source data, they will also be assigned probability based on various smoothing algorithm. Due to the restriction of not accepting the vocabulary, we will not use this method in our task. We will only show the meaning of the setting parameters of this method in the setup file.

Another major restriction for this method is that it only creates bigram models, and the smoothing algorithm is crude (a simple absolute discounting). Thus, this method is often used in phoneme recognition experiments, such as TIMIT. However, we still provide it in the tool as a convenient and quick method. Its settings are as follows. Some parameters are appended by a “1”. This is only to distinguish this method with method 2.

Method1_on: y turns on this method; n turns it off.
Startword1 & Endword1: internally, when processing each sentence, a startword as specified here will be prefixed to the sentence. Similarly, an endword will be appended after each sentence. The startword and endword avoid confusion of counting the last word a sentence and the first word of the next sentence as a bigram. In our example, we use “SENT_START” and “SENT_END” as the startword and endword. Note that these words must be in the dictionary. Make sure they have the following entries:
SENT_START [] sil
SENT_END [] sil

OptString: this string specifies the unigram floor count and the bigram count threshold. A straightforward mathematical form of these parameters can be found in HTKbook, section 17.14.2.
Discount: this is the discount factor in the bigram. Each bigram count will subtract this factor to make room for the unseen events. Its mathematical form can be found in HTKbook, section 17.14.2.
datalist1: this is the only input for the LM, which is a list of all the transcription files. Please copy "LMdata_trs.lst" from "files needed" folder to "lists" folder, and specify the file path here. Then, create a folder called "train_word_trs" inside "data" folder, and copy all the transcription files in the list into this folder. The list "LMdata_trs.lst" contains all transcription files of A, B, C. Since this method cannot account for unseen words, these words will be totally missing from the bigram generated. So, many words in D (which is our test data) cannot be decoded correctly.

There are three output settings:

LM_folder1: specify the folder to store the bigram model generated. This folder will be automatically created.
LM_name1: specify the output bigram model file name (only the file name, not the full path).
wordlist: as a byproduct, a word list consisting of all the words in the LM training data will be output. In our example, the file path is "lists\wordlist_abc." This word list only contains words from A, B, C. Note that the startword and endword are also in the list.

6.5.2. Method 2

Now, let's focus on method2. This method is much more sophisticated than method 1. It supports any n-gram specified by user, and also supports a user-defined vocabulary. All the OOV words are mapped to a unknown class called !!UNK. Any unseen words (which means the word is in the vocabulary but not in the training data) will also be assigned probabilities. So, at least, they can possibly be correctly decoded. From the setup file, it looks that this method is pretty simple. However, there are many intermediate steps in the method, which are hidden from the user. It's recommended that a user should read through HTKbook section 15.1 to 15.3 to learn these intermediate steps. Now, let's look at the parameter settings for this method. For those entries with a "2" appended, the purpose is to distinguish then from method 1.
First, there are some global settings for this method:

method2_on: y turns on method2; n turns it off.
Startword2 & Endword2: these are the start word and end word of each sentence. In this method, the source data format is different from that of method 1. Each row has one sentence. Each sentence starts with the startword, and ends with the endword, as specified by Startword2 and Endword2. These data can come from anywhere, not necessarily from the transcriptions. There is actually an important step called "data preparation" skipped in this manual. A very large amount of cleaning-up needs to be done in this step. For example, punctuation need to be removed, and digital numbers need

to be converted to words. If the source data is downloaded from websites, the headers and hyperlinks are to be removed. In a word, only the pure "language" will be preserved. For Chinese, the character encoding scheme needs to be unified to GB2312, and sentences need to be segmented into words.

This tool assumes that this work has been done. In our task, the LM training data is provided in the file "863data_abc" in the "files needed" folder. Please open it to see the "clean" format of the data. In our example, the startword and endword are still "SENT_START" and "SENT_END." The training data is the transcriptions of A, B and C. In a large vocabulary task, the LM training data usually comes from various sources of human life, depending on the recognition task. The size of the training data easily achieves tens of Gigabytes. For our example task, we will use the transcription data for simplicity.

LM_order: specify the order of the language model. For example, 2 means a bigram; 3 means a trigram. We will first build a bigram model.
DCtype: this is the type of discounting algorithm. HTK provides two types: TG for Good Turing method, and ABS for absolute discounting. Please refer to HTKbook Chapter 14 for details of these two algorithms.
cutoffs: this is the cutoff sequence for the language model. Cut-off is used to throw away those grams who appear infrequently enough. With cut-off, the model size can be greatly reduced, and more frequently observed shorter-context estimates can be made more robust. The cutoff sequence is used to specify a sequence of thresholds. A n-gram that appear more often than its threshold will be preserved when computing the probability estimate; otherwise, this n-gram entry will be thrown away. For example, if the language model order is 3 (a trigram), and the cutoff sequence is 1;1, this means all the bigrams who appear at least two times (greater than 1 time) in the training data will be preserved, so are the trigrams. The rest of them will be discarded. Note that there is no cutoff for a 1-gram (unigram). All the unigrams will be kept. So, the cutoff sequence is only effective when LM_order is at least 2, and when that is the case, the length of the cutoff sequence must equal to LM_order-1, and it is obvious that setting a cutoff factor to 0 means that all of the corresponding n-grams will be preserved. If the LM_order is set to 1, then, the cutoff sequence can be set to any length with any number (since it loses effect for a unigram). In fact, when running this tool, a table will show up on the screen. This table counts how many n-grams will be left when different cutoffs are set. For example, in the following table, it says that a cutoff factor 0 for bigram will make 11181 entries, and a cutoff 1 will have only 1045 entries left. In our example, we set the cutoff to 0, which means we will preserve all the bigram counts.

cutoff	1-g	2-g
0	4488	11181
1	1610	1045
2	936	351

LM_format: specify the format of the language model. The available choices are text format and binary format. In a large vocabulary task, the language model is usually very large. Setting it to "binary" will convert the bigram and any higher order grams to its binary format, and only leave the unigram part in its text format. Thus, the storage size of

the LM will be much smaller. Setting this term to “text” will make the whole file in its text format.
Max_vocab: this term specifies the maximum number of unique words the training data is allowed to have. If the training text contains more unique words than this number, an error will be given.
Next, there are two input files:
datalist2: this is the list of all the training data for the language model. Each file in this list complies with the same format: one sentence in one row, and each sentence starts with the specified startword, and ends with the endword. Please copy the file “LMdata_text.lst” from “files needed” folder to “lists” folder, and put the file path here. Then, create a folder called “LMtext” inside the folder “data,” and copy the training data file “863data_abc” from “files needed” folder to “LMtext” folder.
vocabulary: this is the vocabulary of the LM. Please copy the file “wordlist_abcd” from “files needed” folder to “lists” folder. This list is manually extracted from all the sentences in A, B, C, D. There are many words in group D (which is our recognition task), which are unseen in the training data (A, B, C). But due to the smoothing algorithm, these words will be assigned equal probabilities as unigrams, so that they are still likely to be decoded correctly by the decoder. In many cases, the vocabulary is a subset of the training data. For example, it might be the top 20,000 most frequent words in the training data. In this case, those OOV words (out-of-vocabulary words) will be mapped to a unknown class called !!UNK in the generated LM. If “vocabulary” is set to “none” (case insensitive), this simply means that no vocabulary is provided. In this case, all the words encountered in the training data will be counted as in-vocabulary words, and are used to generated the LM.

Note that the vocabulary provided MUST include the startword and the endword as two entries. In our example, please notice that SENT_START and SENT_END are in the list “wordlist_abcd” in addition to those “real” words in the text. However, the vocabulary must NOT include the entry !!UNK. This is because !!UNK is the unknown class marker to the system, which will be automatically generated, not an input word, since logically, a user cannot input an “unknown” word to the system.

There are two output settings:

LM_folder2: this is the folder to store the output LM, which will be automatically generated if it does not already exist.
LM_name2: this is the output LM file name.

After running this tool, a bigram LM model “bigram2” will be generated inside the “LMs” folder. Please open this file. It’s interesting to notice that the !!UNK entry appears in the unigram. Theoretically, it shouldn’t be there, since all the words in the training data set (A, B, C) are covered by the vocabulary, which is formed by words from A, B, C, D. So, there should not be any unknown words. In fact, internally, when creating a LM, HTK will first assign each new word in the training data a number. All the following operations are based on this mapping. When a vocabulary is passed to HTK, the unknown class !!UNK will be added to this mapping forcefully, regardless of whether there are

OOVs or not (this mapping file=vocabulary+!!UNK). So, since there is actually no !!UNK in the training data, but !!UNK is in the mapping file, it will be regarded as an unseen word, just as those "real" unseen words, which are in set D, but never appear in the training data composed of A, B, C. Thus, !!UNK becomes a unigram and is assigned a probability by the smoothing algorithm.

So, due to this issue, when you know ahead of time that there is no OOV words in the training data, just set "vocabulary" to "none" rather than leaving it there. Apparently, setting "vocabulary" to "none" in this case will result in a more accurate LM, since there will be no nuisance !!UNK, which will affect the estimates of other grams.

As practice, you can change the LM_order and cutoff sequence to create a trigram LM model, or even higher order.

6.5.3. Converting ARPA bigram model to a lattice network

The final step is to convert the ARPA format LM to the lattice format. In the decoding step, we will use two decoders respectively. One is Hvite, and the other one is HDecode. Hvite only works with a word lattice network, not the language model directly. But HDecode works directly with ARPA LMs (both bigram and trigram). So, this step is to prepare for Hvite. There is no need to make this conversion if HDecode is to be used.

For a detailed description on what a lattice network is, please refer to HTKbook section 12.2. The conversion from the ARPA format to the lattice format only works for bigram models. It does not work for any higher order models. Then, the question is: how does the decoder Hvite work with a trigram model, since trigram models can not be converted to the lattice format? Later on, you will know that Hvite will first use the bigram lattice network to do the first round of decoding. Then, another HTK tool HLrescore will be called to expand the output lattice for each sentence to incorporate the trigram and rescore each path in this network. In this processing, the ARPA format trigram will be passed to do the expansion. So, only the bigram model needs to be converted to the lattice format.

With this background, let's look at how the tool implements this conversion. First, a basic point will be made. In our example, as well as the results reported in a later section, we will not use the LMs generated by HTK. Instead, those results are based on LMs generated by SRILM (Stanford Research Institute Language Modelling Toolkit), which is a dedicated toolkit for language modelling. This toolkit provides much more powerful and flexible options in making LMs than HTK. As mentioned before, because of the standard ARPA format, we can easily use LMs generated by other tools other than HTK in the decoder of HTK. Our task is an appropriate example of this kind. The only thing to do is to convert the ARPA bigram model into its lattice format if Hvite is to be used in the next step.

So, first turn off "Train_on" and turn on "Convert_on." Then, please copy the two LM files "bigram_abc" and "trigram_abc" from "files needed" folder to "LMs" folder. These

are the bigram and trigram models made by SRILM. Its training data is also the transcription sets A, B, C, and the vocabulary is also all the words in A, B, C, D. The smoothing algorithm is Witten Bell, which is not an option in HTK LM tools. Please open either of these two LMs. It might be noticed that there is no !!UNK in the LM, whereas in the HTK-based LM, there is. SRILM provides an option to turn off !!UNK. Let's look at the setting parameters of the conversion step:

Startword & Endword: again, please specify the startword and endword in the LM to be converted.

Two input files are needed:

Bigram: this is the bigram file path. In our example, it is set to "LMs\bigram_abc." Recall that the conversion only works for bigram models.
wrdlist: a word list is needed. This word list must cover all the words in the LM, including the startword, endword, and !!UNK (if !!UNK appears in the LM). Please copy the file "wordlist_abcd_unk" from "files needed" folder to "lists" folder, and specify the file path here. Compared with the vocabulary file "wordlist_abcd", a !!UNK entry is added in the last row. In our task, the LM "bigram_abc" does not have a !!UNK term. However, it does not matter that the word list has this extra term in this case, because the only requirement is that the word list needs to fully cover all the words in the LM. Extra words do not affect anything. However, if the LM "bigram2", which is the one generated using method 2 in this tool were to be converted, then, the word list MUST contain !!UNK since this entry did appear in this LM.

A side point needs to be illustrated. As mentioned in method 2, sometimes, we know ahead of time that there are no OOV words in the training data. So, in order to avoid a nuisance !!UNK, we set "vocabulary" to "none." Since we do not need to provide any word list in that step, we might forget to make one for the conversion step. Unlike method 1, method 2 won't automatically generate a word list for us to use in the conversion step. So, when method 2 is selected, a warning will be given, reminding the user that a word list needs to be manually prepared whether a vocabulary is necessary or not.

There are two output settings:

Network_folder: specify the folder to store the output lattice file.
Network_name: specify the lattice file name.

As specified in the setup file, a folder "Networks" will be automatically generated, inside which the lattice file "network" will be stored. This file will be used in the Hvite decoding step.

As practice, please convert "bigram2," which is the bigram generated by method 2 into its lattice format. If you do so, you might notice that in the output lattice file, there will be no !!UNK term, though !!UNK indeed is in the bigram model, as well as in the word list

of the conversion step. This is because the decoder Hvite requires that all the words in the lattice network must have at least a pronunciation entry in the dictionary. However, what should the pronunciation of the !!UNK be? Some people use silence (sil) as its pronunciation. This solution does make the decoder work. But it does not make sense. Logically, !!UNK means a class of words not known to the system. They are out of the vocabulary, which defines all the words capable of being recognized by the system. Then, what should be the pronunciation of a word unknown to the system? No one actually knows. So, in this tool, we do not have an entry !!UNK in the pronunciation dictionary, since the system does not know its pronunciation. But to make Hvite work, we have to delete !!UNK from the lattice during the conversion. The other decoder HDecode does not have this problem. It is able to deal with !!UNK in a LM, and it works directly with the ARPA LM format. Also, there is no need to have the !!UNK entry in the pronunciation dictionary, which makes logical sense.

6.6. Decoding (Tool_Decode.m)

With the acoustic and language models at hand, we are ready for the final step: decoding. The tool for the decoding step is “Tool_Decode.m,” and its setup file is “Tool_Decode.dcf.” Please copy these two files from “Tools” folder to “exp” folder.

This tool provides two decoders: Hvite and HDecode. Hvite is suitable for small and medium size vocabulary systems, and works better for monophone and internal word triphones. It becomes progressively inefficient as the size of the vocabulary grows and cross word triphones are used. HDecode is a dedicated decoder for large vocabulary systems. It only works with cross word triphones, which is the typical acoustic model type for any large vocabulary system. HDecode is much more efficient (much lower real time factor) than Hvite when cross word triphones are used in both decoders.

Please open the setup file "Tool_Decode.dcf." Let's first look at the global settings for both decoders.

Trace_on: 'y' displays the progress on the screen; 'n' turns it off.
Clean_up: 'y' cleans up the old output MLF file (for both decoders), as well as the output lattice file (for Hvite) before new ones are generated. 'n' turns it off.
LogDir: a progress report "progress_decode.log" will be generated in this directory. Note that the percentage accuracy is also written to this report.
Decode_on: 'y' turns on decoding. 'n' turns it off.
Feat_list: this is the feature file list path. It contains all the feature files to be decoded. Both decoders need this list.
Feat_folder: this is the feature file folder. This entry is only for the use of Hvite. Along with the decoded MLF file, a lattice network for each feature file will also be generated. These lattice files will be used in decoding with a trigram (Hvite works directly with a bigram only. To use a trigram, a lattice network based on the bigram decoding must be generated first). They are placed in the same folder as the features. So, the tool will copy these lattices from the feature file folder specified by "Feat_folder" to another folder called "Lattice_folder" (specified later) . When the trigram decoding is turned on, lattice

files will be loaded from "Lattice_folder" (the original ones in "Feat_folder" will be deleted). The other decoder HDecode works directly with a trigram model. So, there is no need to specify the "Feat_folder" for copying lattices.
Startword & Endword: these are the startword and endword of each sentence in the language model. They are exactly the same as those specified in the LM generation step. These sentence start and end tokens need to be specified for both decoders. Note that in the dictionary, there must be two entries for these words. In our example, they are:
SENT_START [] sil
SENT_END [] sil
Result_folder: this is the folder to store the output MLF file. Note that the percentage accuracy will be written the progress report "progress_decode.log." The folder specified in the entry only stores the output MLF file.
Test_trslist: this is a list of all the ground truth transcriptions of the test data for the computation of the recognition accuracy. Please copy the file "test_wordtrs.lst" from "files needed" folder to "lists" folder and specify the path here. Then, create a folder "test_word_trs" inside "data" folder, and copy all the transcriptions of the test data into this folder. The transcription must be in its raw format: one transcription for one sentence, and one word in each row. The tool will first convert the raw format to its MLF format. In the conversion, each Chinese word will be split into characters, since people compute the character level accuracy in a Chinese ASR system. So, in the MLF file, there is one Chinese character in one row. If the task language is English, this "splitting" will lose effect. So, an English word is still an English word. There is no need to specify which language we are recognizing.

Next, let's look at the first decoder: Hvite

6.6.1. Hvite

Before we get into the settings, there are some general properties and restrictions of this decoder:
a. Hvite supports decoding with monophone, internal word triphone, and cross word triphone.
b. Hvite supports decoding with bigram LM directly. The LM must be in its lattice format.
c. To use a trigram, the output lattice for each feature file is expanded by the trigram, and each path in the expanded network is rescored. The path with the highest score is selected as the output. So, the trigram decoding is based on the result of the bigram decoding. The trigram is in its original ARPA format.
d. As mentioned in the language model generation part, Hvite does not work with the unknown word class !!UNK. So, when converting the bigram model to its lattice format in the language model tool, the !!UNK is deleted.

Now, let's look at the settings for Hvite (in addition to the global settings for both decoders).
Hvite_on: 'y' turns on Hvite. 'n' turns it off.

HMM_type: this is the underlying acoustic model type. 'iwd' for internal word triphones; 'xwd' for cross word triphones; 'mono' for monophones. In our example, this is set to 'iwd'.
Dict_hvite: specify the dictionary file path. The word lattice network will be expanded into the underlying phoneme network by looking up this dictionary. So, the dictionary must contain all the words in the language model, including the sentence startword and endword. Please create a folder "dicts" inside "exp" folder, and copy the file "dict863_tone_sp1" from "files needed" into "dicts" if you have not done so in the triphone training step, and specify the file path here.
HMM_folder_hvite: this is the directory to load all HMM models. In our example, the HMMs are stored in "hmms\fhmmtri_inword."
HMM_list_hvite: this is the list of all the HMMs. In our example, it is set to "lists\tiedlist_inword."
Conf_iwd, Conf_xwd & Conf_mono: these are the configuration files for decoding with internal word triphones, cross word triphones, and monophones. Please copy the files "hvite_iwd.conf," "hvite_xwd.conf" and "hvite_mono.conf" from "files needed" folder to "toolconfs" folder, and specify the paths in the corresponding places. The tool will locate which file to use according to the "HMM_type" specified above. Note that you do not need to specify all three files at the same time, because only one HMM_type is used in each decoding processing. You can leave the other two places anything but empty, such as 'none.'
HviteOptstring: this string controls related parameters for Hvite. Frequently used ones are -t, -s, -p, -u, -v. In our example, '-t' is the pruning factor. It greatly affects the speed and accuracy of the decoder. Larger -t value leads to higher accuracy, but lower speed. Typical values for -t are between 200 to 250. '-s' is the language model scale factor, which also has significant impact on the accuracy of the decoder. Normally speaking, its value is affected by the size of the vocabulary as well as the size of the HMM set. Typical values for a large vocabulary system are 12-15. Larger size of the vocabulary and HMM set leads to large value of -s. '-p' is the word insertion penalty factor. Normally, its value is fixed at 0. You can add -u and -v in the string to see what impact they will have on the accuracy. For the detailed meaning of these options, please refer to HTKbook section 17.23.

The followings are for bigram decoding.

Bigram_on: 'y' enables decoding with a bigram. 'n' turns it off.
Network: specify the bigram lattice network file path as an input. As described above, Hvite only works with bigram lattice format, not ARPA format.
Rec_output_bg: specify the file name for the decoded MLF using a bigram. Note that only specify the file name here, not the full path. This file will be generated in the directory specified in "Result_folder."
Lattice_folder: specify the folder to store the output lattice for each feature file. These lattices will be used in the trigram decoding. In our example, it is set to be "..\data\Lattice." A folder "Lattice" will be automatically created inside "data" folder.
Lattice_list: specify the file path for a list of all the lattices. This is an output. This list will be used in the trigram decoding.

The followings are trigram decoding settings:

Trigram_on: 'y' turns on trigram decoding; 'n' turns it off.
Trigram: specify the trigram file path. In our example, it is set to "LMs\trigram_abc." As described in the language model task, this trigram is built from training data A, B, C. But all the words in A, B, C, D are put in the vocabulary. Note that this trigram is in ARPA format, not in lattice format.
Latlist: this is the file path for the list of all the lattices generated in the bigram decoding step. Each lattice network in this list will be expanded by the trigram, and each path will be rescored (only the language model score, the acoustic score remain unchanged). The path with the highest score in a network will be selected as the output sequence for the trigram decoding.
Conf_rescore: this is the configuration file path for the trigram decoding. This configuration file will be automatically generated in the path specified.
HLrescore_Optstring: this string has the same meaning as the "HviteOptstring." For a detailed list of options, please refer to HTKbook section 17.13.
Rec_output_tg: specify the output MLF file name for the trigram decoding. Again, this is only the file name. The file will be generated inside the folder specified by "Result_folder."

Next, let's look at the other decoder: HDecode. This decoder is not included in the regular HTK package. An additional, more restrictive license must be agreed in order to download HDecode. HDecode can be downloaded from the official website of HTK. This manual assumes that it has been correctly installed.

Again, let's first look at the general properties and restrictions for this decoder.
a. HDecode is designed for large vocabulary task. It ONLY works with cross word triphones.
b. HDecode works directly with bigram and trigram models. The LMs are in the ARPA format.
c. sil and sp models are reserved as silence models. sil must be used as the pronunciation for the sentence start and sentence end tokens in the dictionary. sp is the short pause between words. sp is automatically added to the end of all pronunciation variants of each word in the recognition dictionary. So, each word in the dictionary MUST NOT have a sp appended.
d. Only the sentence start and end tokens (SENT_START, SENT_END in our example) are allowed to have sil as their pronunciations. sil can NOT appear anywhere else.
e. In the HMM set, only sil and sp are allowed to be monophones. Others must be cross word triphones.
f. HDecode works with unknown class !!UNK. So, the LMs are allowed to have !!UNK entries. !!UNK should not appear in the dictionary.

As stated in the Hvite section, Hvite also works with cross word triphones. However, Hvite is much less efficient than HDecode, especially for a large vocabulary task. Though Hvite has very close accuracy as HDecode, HDecode has a much lower real time factor.

So, HDecode is strongly recommended for large vocabulary tasks with cross word triphone models.

Now, let's look at the settings for HDecode in the tool (in addition to the global settings for both decoders).
HDecode_on: 'y' turns on HDecode. 'n' turns it off. Note that HDecode and Hvite can not be turned on at the same time. Only one decoder is allowed in one decoding processing.
Conf_hdecode: this is the configuration file path for HDecode. This file will be automatically generated.
HdecodeOptstring: this is the operation string of HDecode. The options in this string have the same meaning as those in Hvite. Please refer to HTKbook, section 17.6 for details of all the available options.
Use_bigram: 'y' turns on bigram decoding. 'n' turns it off. Note that HDecode works directly with bigram and trigram models. So, there is no need to run bigram decoding first in order to use a trigram model.
Use_trigram: 'y' turns on trigram decoding. 'n' turns it off.
HMM_folder_hd: this is the folder to load HMMs. In our example, all the cross word triphones are stored in "hmms\fhmmtri_xwd."
HMM_list_hd: this is the list of all HMMs to be loaded. In our example, the file path is set to "lists\tiedlist_xwd."
LM_bigram & LM_trigram: these are the file paths for LMs. There should be two files "bigram_abc" and "trigram_abc" in the folder "LMs." HDecode works directly with the ARPA format, not the lattice format. Note that there is no need to specify the paths for both LMs. Depending on which order of LM is to be used (set by Use_bigram and Use_trigram), only the corresponding LM file path needs to be specified. The other place can be set to anything but empty, such as 'none' or '\'.
Dict_hd: this is the dictionary file path for HDecode. Please copy the file "dict863_tone_nosp" from "files needed" folder to "dicts" folder and specify the path here. In this dictionary, all the sp's have been removed, and silence pronunciation (sil) only occurs in the sentence start and end tokens. Note that if your dictionary is copied from the forced alignment step, there should be an entry "SENT_Boundary [] sil." Please remove this entry to make sure that sil only occurs in the sentence start and end entries.
Rec_hd_bg: this is the output MLF file name for bigram decoding. Again, this is only the file name. This file will be stored in the folder specified by "Result_folder" in the global settings.
Rec_hd_tg: similarly, this is the output MLF file name for trigram decoding.

At this point, we've completed all the steps in this ASR system. The percentage accuracy for the test data can be found in the progress report "progress_decode.log."

6.7. Experimental results

In this section, a series of experiments are conducted using this ASR toolbox. The database is the 863 Mandarin Chinese database. 78 women speakers from this database were used to train acoustic models. There are 2185 training sentences, which are divided into 4 groups: A, B, C, D, and each group of sentences are spoken by multiple speakers,

resulting in 37116 utterances. 4 speakers from group D are included in the training data. The test data is formed by another 5 speakers from group D, resulting in 3125 utterances. For the language model, the training data consists of the transcriptions of A, B, C. But we put all the words in A, B, C, D into the vocabulary. There are many words in D that are unseen in A, B, C. So, according to the smoothing algorithm, these words are assigned equal probabilities as unigrams.

In Table 17, we present the Chinese character level percentage accuracy results using tonal phone acoustic models. All the models are 16-mixture models. Two feature types are used: baseline MFCC method, and the spectral/temporal method. The baseline MFCC uses frame length 25ms, and frame space 10ms, and has 39 features (12 DCTC coefficients+log energy+delta and acceleration terms). The spectral/temporal method uses 13 DCTC and 6 DCS (78 features), frame length 25ms, frame space 2ms, and block length 102ms (51 frames), block space 12ms (6 frames). As stated before, in forming triphones, the TB factor controls the degree of state tying, thus has great influence on the accuracy. The other factor used in state tying called 'RO' is fixed at 100. Also, in the decoding stage, the language model scale factor '-s' can affect the accuracy significantly. So, in Table 16, we also list the optimal values for TB and s wherever applicable. A bigram model is used for all the results in Table 16.

Table 16: Results for tonal phone acoustic models, LM=bigram

	Monophone	Internal word triphone	Cross word triphone
MFCC	80.3%	84.7%	85.5%
FFT+DCTC/DCS	82.4%	86.1%	87.5%
TB value	None	2000 for both methods	2000 for both methods
-s value	10 for both methods	12 for both methods	12 for MFCC, 15 for FFT+DCTC/DCS

In Table 17, we present the same results using base phone acoustic models. The models are 16-mixture models. The feature types are the same as in the previous experiments. The baseline MFCC still uses 25ms frame length, and 10ms frame space, 39 features. The spectral/temporal method uses 78 features (13 DCTC/DCS) as before. The frame length is still 25ms, frame space 2ms, black length 102ms (51 frames); but the optimal block space changes to 14ms (7 frames). Again, the same bigram LM is used in this group of experiments.

Table 17: Results for basephone acoustic models, LM=bigram

	Monophone	Internal word triphone	Cross word triphone
MFCC	75.9%	81.5%	82.5%
FFT+DCTC/DCS	77.6%	82.7%	84.2%
TB value	None	2500 for both methods	2500 for both methods
-s value	7 for both methods	10 for both methods	10 for both methods

In Figure 31, we compare the character level accuracy using tonal monophone and cross-word triphone. The feature used is 42 MFCC including pitch (39 MFCC features+3 pitch features). The number of HMM mixtures increases from 1 mixture to 32 mixtures. The triphone performance is significantly better than monophone.

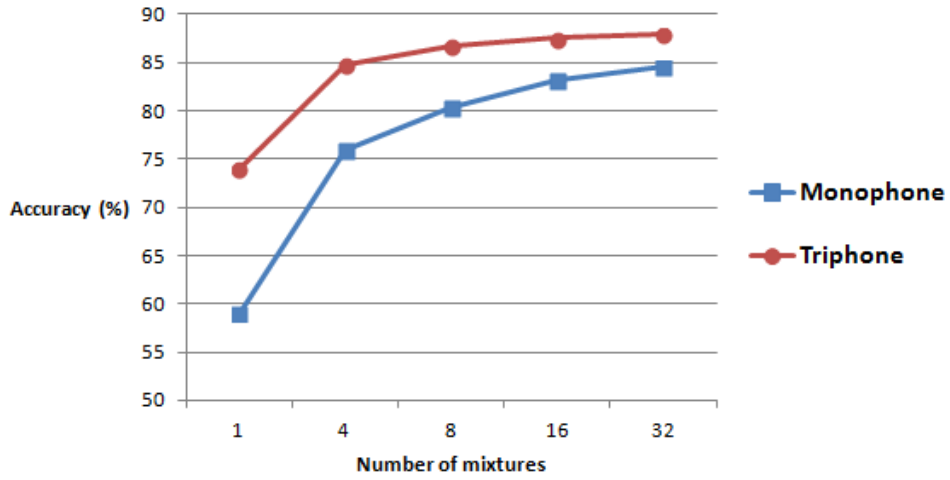

Figure 31: Character accuracy using tonal monophone and triphone models versus number of mixtures.

In Table 18, we compare three different pitch trackers: Yaapt, Yin and Praat. The acoustic models are 16-mixture tonal monophones. The baseline is still MFCC method (39 features without pitch). Pitch features are used in the comparison, resulting in 42 features (39 MFCC features+3 pitch features). Pitch is normalized by the mean and standard deviation of the whole sentence, and its delta and acceleration terms are incorporated. All voiced mode is used in the computation of pitch. The language model is still the same bigram model. The optimal '-s' value is 10 both all of them.

Table 18: Results using different pitch trackers, LM=bigram

	Tonal monophone
MFCC	80.3%
MFCC+Yaapt	83.1%
MFCC+Yin	82.5%
MFCC+Praat	82.7%

Finally, in Table 19, we compare the results using bigram and trigram models. Similar to the bigram model, the trigram is built out of the training data A, B, C, and has all the words in A, B, C, D in the vocabulary. The acoustic models used are tonal internal word triphones and cross word triphones. The feature type used is 78 spectral/temporal features as in Table 16. The optimal TB and -s values are also listed in Table 16. The trigram -s value is the same as that of the bigram.

Table 19: Accuracy using bigram and trigram models for tonal triphones

	Bigram	Trigram
Internal word triphone	86.1%	86.3%
Cross word triphone	87.5%	87.6%

As expected, the trigram model does not help much to improve the overall accuracy. This is because many words in the test data are missing in the training data. So, these unseen words only exist in unigram format, not trigram. Perplexity can be used to evaluate two language models with the same vocabulary on the same test data set. Lower perplexity means a better language model. The LM evaluation is not included in the ASR toolbox. Please refer to HTKbook section 15.4 for details. As practice, you can use the HTK tool LPlex to compute the perplexity of the bigram and trigram models used in our task example. It will be found that their perplexity on the same test data set is almost the same.

7. FORCED ALIGNMENT TOOL USER GUIDE

7.1. Overview

This package is designed for finding the correct pronunciation transcriptions for a large dataset. Usually, a large database does not provide phone level transcriptions. The phone level transcriptions, however, are needed for training HMM models. To find the correct phone level transcriptions, the basic assumption is that a database of wave files with associated word level transcriptions for each sentence is already available. A pronunciation dictionary for each word in the database is also needed. In the dictionary, some words might have multiple pronunciations. So, these words should have multiple entries. If a long (several minute) speech passage, with transcription is available, this long passage should first be segmented into short sentence level (typically around 5 to 10 seconds each) segments with associated text transcriptions for each sentence, in order for the forced alignment and training to work well. If phonetic level transcriptions are already available, (i.e., TIMIT), this step is not needed at all, since essentially this step is only needed to create the phonetic level transcriptions. When dealing with a large database, the training stage of an ASR system involves two steps: initialization and embedded training. In initialization, usually all the models will be initialized by the global mean and covariance of all the acoustic features in the same manner. After initialization, the next step is embedded training. HTK will connect the models embedded in each sentence according to its phone transcription, and accumulate the statistics such as the mean and covariance, using the Baum-Welch method. Once all the data in the database has been processed, the accumulated statistics will be used to re-estimate the model parameters for all the models simultaneously. Since the model connection in embedded training depends entirely on the phonetic transcription, and embedded training is the core step in the training stage, it is very important to insure correct phonetic transcriptions using forced alignment. Note that the embedded training step does not need or use time markers at the phonetic level

Generally speaking, the forced alignment task includes three steps: feature extraction, training an initial set of models, and forced alignment. Before these three steps, the starting point is a set of wave files, along with a set of word level transcriptions, as stated above. After the features have been extracted for each wave file, the word level transcription of each sentence will be expanded to an initial phonetic transcription using the pronunciation dictionary. For words that have multiple pronunciations, this initial phonetic transcription will arbitrarily use the first pronunciation, regardless of the true pronunciation by the speaker. This is why forced alignment is needed. Then, an initial set of HMM models will be trained using this set of initial phonetic transcriptions. The training involves two steps: flat start initialization and embedded training as described above. Finally, this initial set of models will be used to find the correct phonetic transcription of each sentence based on the features, namely, what the speaker really pronounced. Meanwhile, the updated phonetic transcriptions are used to retrain the HMM models. This processing will be iterated a few times. After the last iteration of forced alignment, the resulting phonetic transcription will be regarded as the "perfect" version, and will be used in all the other parts of an ASR system.

The forced alignment tool package includes three separate tools:
1. Tool_Compute_Feat: this tool is used to extract features from a list of wave files.
2. Tool_trainFA: this tool is used to train an initial set of HMM models.
3. Tool_FA: this tool is used to do forced alignment and retrain the HMM models.

Note that tools 1 and 2 can be used for tasks other than forced alignment. However, before tool 3 can be used, features files must be created and an initial set of HMM models are needed—meaning that either tool 1 and tool 2 must be used first, or the features and acoustic models must be created with some other similar tools, which use the same file formats as tools 1 and 2. Note that the forced alignment tool, in addition to creating the phonetic transcriptions, updates HMM acoustic models.

Each tool is a matlab m file, and each m name begins with “Tool_...” A setup file is needed for each tool, and the default file name of the setup file is always the same as the matlab file name, except the file extension is “dcf” rather than “m.” For example, the default setup file for Tool_trainFA.m is Tool_trainFA.dcf. The setup file for a tool contains all the control options and parameters for that tool. When a tool is called, the setup file is the only argument that can be passed. For example, to call Tool_trainFA.m, the format will be Tool_trainFA(‘Tool_trainFA.dcf’). The setup file and the matlab file should be placed in the same experiment folder. If no argument is provided for a tool, the default setup file is assumed (same name as the tool name, except extension “.dcf.”) Next, a detailed description of how to use each tool is given.

7.2. Tool_Compute_Feat

7.2.1. Data preparation

Before you run this tool, some data preparation needs to be done. First, create a folder called “data.” Inside this folder, create a subfolder called “train_wave.” This is the folder

where all the training wave files are to be placed. Copy all the training wave files into this folder. Next, go outside “data” folder, and create another folder called “exp.” This will be your experiment folder. Then, copy "Tool_Compute_Feat.m" and "Tool_Compute_Feat.dcf" from the folder "Tool_FA" into the “exp” folder. In addition, also copy "readhtk.m" file from "files needed" folder into "exp."

In addition, you need to make a list of all the wave files to be processed. You can easily write a program to do this. A short program called “makelist.m” is provided to do this (copy this program from "files needed" folder into “exp” folder). In this program, you need to specify which folder the wave files are stored (“..\data\train_wave” in this case). The path is with respect to the experiment path. After you run the program, a folder “lists” will be created, inside which a list file“wavefile.lst” is created (These names can be changed easily in this program). This list will look like this with each file in one row:

..\data\train_wave\F00C1041.WAV
..\data\train_wave\F00C1042.WAV
..\data\train_wave\F00C1043.WAV
..\data\train_wave\F00C1044.WAV
..\data\train_wave\F00C1045.WAV
..\data\train_wave\F00C1046.WAV
..\data\train_wave\F00C1047.WAV

After the steps above have been completed, and the dcf file is edited as explained beow, the tool can be run to extract features.

7.2.2. Run tool

Use a text editor to open the file "Tool_Compute_Feat.dcf" inside "exp" folder. First, you will see some control questions such as "Trace_on," "Feat_on," etc. These are the options to turn on/off these functionalities. The only two valid answers for such questions are "y" and "n" (case insensitive). There are some other files or folders that you need to specify for the tool in this setup file. We describe each of them later in this document. Also, you can indicate comment lines using "%" at the beginning of the line. A comment line does not have to end with another "%. For clarity, "%" can be placed at each end.

Make sure that the setup file does not contain any "tab" (\t). "\t" is not recognized by the tool code.

To begin with, in the dcf setup file, there is a section called "Global settings." It has three items:

Trace_on: if you set "y," you will be able to see the progress of the feature extraction for each utterance on your screen. You can turn it off by setting "n."
Clean_up : "y" means to clean up (i.e., remove) all the old feature files in your feature file folder (specified later) before feature extraction.

LogDir: you can specify a folder to store a progress report. The report is named as "progress_feat.log." This report will keep track of whether an error happened, how many features for each wave file, and what kinds of features were extracted, etc.
Next, there are two questions:
Feat_On: "y" means to turn on feature extraction. "n" means to turn it off.
FrtEnd_opt: this is to select different feature extraction methods. There are three available options: HTK_MFCC, HTK_PLP and User. HTK_MFCC and HTK_PLP are provided by HTK, and "User" means the tfrontm frontend. These options are all case-insensitive. If you choose to use tfrontm frontend, there are some extra steps to follow before you use it.
a. Copy the folder "v7" in "files needed" to "exp."
b. Open a command window, compile "tfrontm.m" inside "v7" using "mcc" command, then create a folder called "tfront" inside "exp" folder, and copy the file "tfrontm.exe" from "v7" to "tfront." Note that you need to have a “C” compiler installed (such as Microsoft visual C). The version of the C compiler (64 or 32 bit) needs to match that of your matlab. The Matlab compiler toolbox is also needed.
c. Copy the file "cp_42.ini" from "files needed" folder to "tfront" folder, and rename the file "cp_fea13.ini".
d. Copy the file "SNR801.trn" from "files needed" folder to "tfront" folder, and rename it "tfrontm.dat."
Please refer to a detailed description of the tfrontm frontend if you want to use it.

Note that no matter which frontend you use, the wave files are always needed. In tfrontm frontend, we provide a function called "rd_audio.m." The wave file format supported by this function includes: NIST (which is used by TIMIT, also called SPHERE), WAVE (also called RIFF, which is the Microsoft WAVE files used on PCs), and RAW (which has no headers).
Wave_List: you need to specify the wave file list that contains all the wave files to be processed. In the example, "lists\wavefile.lst" is provided. This was generated by the program "makelist.m." This list, and a procedure for making the list, was described above.

Next, note that no matter which frontend you choose, two things will be generated:
Feat_folder: this is the folder that stores all the feature files generated. You do not need to create this folder manually. It will be automatically created. You only need to specify its path. In our example, the path is "..\data\Feat." Again, the path is with respect to the experiment folder, which is "exp."
Feat_List: A list of all the feature files will also be generated for later use. This is where you specify the path of this list. In our example, as specified, a "featfile.lst" will be generated in the folder "lists."

Next, we come to the "HTK_MFCC" settings. First, create a folder called "toolconfs" inside the "exp" folder, and copy the file "hcopy_MFCC.conf" from "files_needed" folder into "toolconfs." "hcopy_MFCC.conf" is a configuration file for HTK_MFCC frontend. It specifies a set of necessary parameters for this frontend, such as the frame length, frame

space, wave file format, number of filterbank channels, etc. Please read Chapter 5 of the HTKbook for details. Then, specify the configuration file path in "Conf_MFCC."

Note that in our example, the wave file format is WAVE, which needs to be known ahead. So, as you can see in "hcopy_MFCC.conf," the "SOURCEFORMAT" is also set to "WAV" to match the wave file format. If you change to TIMIT database, you need to change "SOURCEFORMAT" to "NIST" because that is the wave file format in TIMIT. Chapter 5 of the HTKbook provides details of all the wave file format supported by HTK.
Conf_MFCC: specify the configuration file for HTK_MFCC frontend.

Similarly, if you want to use HTK_PLP frontend, you need to copy the file "hcopy_PLP.conf" from "files_needed" folder into "toolconfs" and specify its location in "Conf_PLP." Also, change "FrtEnd_opt" to HTK_PLP
Conf_PLP: specify the configuration file for HTK_PLP frontend.

Finally, if you want to use the user defined tfrontm frontend, you need to specify its configuration file as well in "Conf_tfrontm." Please refer to our tfrontm manual for detailed description for this frontend.
Conf_tfrontm: specify the configuration file for tfrontm. As described before, the location of this file is tfront\tfrontm.dat.

You do not need to specify the configuration files for all three frontends since you will only use one of them at a time. The tool will only check the file location according to which frontend you selected. However, do NOT leave any configuration file place blank. For example, if you choose "User" as your frontend, you can put a "\" or "N\A", or "None" (or anything meaningful to you) in Conf_MFCC and Conf_PLP, and only specify the configuration file in Conf_tfrontm. But do NOT leave Conf_MFCC and Conf_PLP blank.

7.3. Tool_trainFA

This tool is to train a set of initial acoustic models for forced alignment.

7.3.1. Data preparation

Before you run this tool, you need to prepare a set of word level transcriptions for the wave files. Usually, when you deal with a large database, you do not have the phone level transcriptions; you only have the word level transcriptions for the utterances. The "standard" word level transcriptions HTK can use have one word in each row, and one transcription file for each sentence. Note that the file name of a transcription file must be identical to the file name of the feature file for that sentence (However, the extensions can be anything). Since there are many possible formats that the original word transcriptions can have when you obtain the database, there is no "standard" code to convert it to the HTK acceptable format. You need to write your own code to do this step. In our example, you need to create a folder called "train_word_trs" inside the folder

"data," and put all the "standard" word level transcriptions you made into this folder. To make it clear, you can set the extension of the transcription files to be ".WRD."

Also, you need a list of all the transcription files. You can make this list using "makelist.m". You need to change the "path_wave" variable to "..\data\train_word_trs," and change "listfile" to "wordtrs.lst," and run the program. A "wordtrs.lst" will be generated in the "lists" folder.

Copy "Tool_trainFA.m" and "Tool_trainFA.dcf" from "Tool_FA" folder into "exp" folder.

7.3.2. Run tool

Use a text editor to open "Tool_trainFA.dcf". Again, you will see the same three global settings. "Trace_on" has the same meaning as before. But "Clean_up" has different meaning:

Clean_up: if this is set to "y", each target HMM folder (specified in later steps) will be cleaned up before new models are generated, if there are any old models in those folders. By “clean up” we mean that existing models will be deleted.
LogDir: again, you can specify a folder to store a progress report for the training processing. The default report file name is "progress_trainFA.log." In this report, some information will be provided, such as the number of mixtures, the number of states, the progress of mixture splitting, and the feature vector length, etc.

The next step is "transcription preparation." In the very beginning, you have created a set of word level transcriptions for all the utterances, and a list of all these transcription files as well. In HTK, what is used is the "MLF" format of these transcriptions. Basically, a MLF file puts all the transcriptions in one file, and prefixes each file by its path. "transcription preparation" step converts the original word level transcriptions to its MLF format, and also generates initial phone transcriptions based on the word MLF file.
Trans_prep: "y" turns on transcription preparation. "n" turns it off.
Gen_Word_MLF: "y" converts the original word level transcriptions to a MLF file.
Word_trs_list: you need to provide a list of all the original word transcriptions to convert. In our example, it is lists\wordtrs.lst.
Conf_wrdmlf: this is the configuration file to make the MLF file. Copy "wordmlf.led" file from "files needed folder" to "toolconfs" folder, and specify the file path here.
WordMLF: this is the output MLF file. You need to manually create a folder called "labs" inside "exp" folder, and a MLF file "word.mlf" will be generated inside "labs" as specified in WordMLF place.

After the word MLF file has been generated, the next step is to create two phone level MLF files. The first MLF file does not contain "sp," which is a short pause after each word; the second MLF file has a "sp" after each word. Later on, a set of low order HMM models (usually 1 or 2 mixtures) that does not have "sp" model will be trained using the first phone level MLF file, and then, a "sp" will be introduced, and the second MLF will

be used to train this expanded set of HMM models. So, here, we need two MLF files for later use.
Gen_Phn_MLF: "y" means to generate phone level transcriptions. "n" turns it off.

Four input files are needed to generate phone level transcriptions:
WordMLF: this is the word MLF generated in the last step. Put the same file path here as in WordMLF.
Dict: a dictionary file is needed to convert word MLF to phone MLF. Manually create a folder called "dicts" inside the "exp" folder, and copy the file "dict863_tone_sp1" from "files needed" folder into "dicts" folder. Then, specify the dictionary file path in Dict. If you open this file, you will see that all the words are placed on the left side, and their phone pronunciations are on the right. Many words have multiple pronunciations. Note that a "sp" is appended after each word. There is an entry "SENT_Boundary," and its pronunciation is "sil." This entry is for the forced alignment step, where the "SENT_Boundary" will be added before and after each sentence.
Conf_wrd2phn_nosp: this is the configuration file to convert the word MLF to the phone MLF without "sp." Copy the file "word2phn_nosp.led" from "files needed" folder into "toolconfs" folder, and specify the path here.
Conf_wrd2phn_sp: this is the configuration file to convert the word MLF to the phone MLF with "sp." Copy the file "word2phn_sp.led" from "files needed" folder into "toolconfs" folder, and specify the path.

If you open either "word2phn_sp.led" or "word2phn_nosp.led", you will see a couple of commands, one command in each row. For example, "EX" means to expand each word with its phone pronunciation; "IS sil sil" means to insert a "sil" before and after each sentence. For a complete set of command descriptions, please refer to HTKbook section 17.10.

Four output files will be generated:
PhoneMLF_nosp: this is the phone MLF file without "sp." In this example, it is generated in "lab" folder, and its file name is phone_nosp.mlf.
PhoneMLF_sp: this is the phone MLF file with "sp" between words. In this example, its file path is labs\phone_sp.mlf.

Note that when HTK expands each word, only the first pronunciation of each word in the dictionary will be used. So, both the phone MLF files contain only the first pronunciation of each word. These initial MLF's will be used to train an initial set of HMMs. Then, forced alignment will be used to find the correct pronunciations according to the acoustic information.
PhoneList_nosp: a list of all the phones encountered will be generated. This list does not contain "sp." it will be placed in the "lists" folder, and its file name is "monophone_nosp" as specified.
PhoneList_sp: a list of all the phones encountered will be generated. This list contains "sp." Specify its location in a similar way.

At this point, the transcription preparation step is done. These four output files will be used in the training step. Note that if you have already done this step, you can simply turn off "Trans_prep."

The next step is to train a set of HMM models for forced alignment. Similarly, there are some global settings to be specified at the beginning.
Train_on: "y" turns on training. "n" turns it off.
Feat_List: this is the feature file list generated by the feature extraction tool. In Tool_Compute_Feat.dcf, it is specified in "Feat_List" place. Put the same file path here for training.
Feat_List_ini: HTK computes the global mean and variances of all feature files provided to initialize each HMM model. However, sometimes, this will be a poor initialization for the following step, which is called embedded training. Empirically, do not use short sentences to initialize the models. If many short sentences (1 or 2 seconds long) are used for initialization, the embedded training step will fail. You should use long sentences, at least 3 seconds long each. So, you can select a subset of all utterances, and make a list of them, then, use this list of feature files to initialize the models. In our example, this list is located as "lists\featfile_ini.lst." After the models have been initialized, the full feature file list "Feat_List" will be used to do the embedded training.
numState: the number of emitting states. Two non-emitting states will also be used (one before and one after the emitting states). For example, if numStates=3, then, there are actually 5 states.
numMixture: specify the mixture splitting sequence. The models usually start from a low order (such as 1 mixture), and then gradually split to the desired order, each time increasing by a small amount. Each splitting step is separated by a ";." For example, "1;2;4;6;8;12;16" means to start from 1 mixture models, and split to 2 mixture models, and then 4, 6, 8, 12, 16. You can directly start from a high order model, such as 16, but usually won't get robust training. It's strongly recommended that you go through this splitting processing. You can refer to HTKbook section 10.6 and section 17.8 for details of how the splitting works and why it is more robust than directly starting from a high order.
Iteration: this specifies how many iterations of training will be conducted after each splitting. Again, this sequence is separated by ";" and it must have the same length as the mixture splitting sequence. For example, "3;5;6;7;7;7;7" means 3 iterations of training for 1 mixture models, 5 for 2 mixture models, etc, if numMixture is set to be "1;2;4;6;8;12;16."
Final_hmmfolder: this is for convenience. In our example, this folder path is set to be "hmms\fhmm." These folders (hmms and fhmm) will be created automatically inside "exp" folder. There are two steps in model training: initialization and embedded training. You will notice that for each step, there is a "target folder" in the setup file to store the HMMs after each step (hmm1 for initialization and hmm2 for embedded training in our example). However, you may forget which step you stop at in training. So, in the forced alignment stage, when loading the HMMs, you do not know where to load the HMMs. Hence, a final HMM folder needs to be specified to store the final HMMs, no matter which training step you stop at. That means: if you choose to do the embedded training, the models from hmm2 (which is the target folder for embedded training) will be copied

to the final HMM folder; if you stop at initialization, the models from hmm1 (which is the target folder for initialization) will be copied to the final HMM folder. Hence, you can always load HMMs from the final HMM folder in the forced alignment step. In addition, if you want to modify the previously trained HMMs, such as adding more mixtures by splitting, or adding more training iterations, you can directly load HMMs from the final HMM folder without memorizing which step you were at when you stopped last time.

Next, we come to the initialization settings.
Init: "y" turns on initialization; "n" turns it off.

Then, 4 input files/folders are needed for initialization.
hmmList_nosp: this is the list of HMMs generated by the transcription preparation step. This list does not include "sp" model. Later on, a "sp" model will be introduced after the first mixture model (1 mixture in our example) is trained.
SrcDir_init: this is the source directory for the prototype HMM to be initialized. In HTK, before initialization, a prototype HMM needs to be defined. Basically, the prototype defines the topology of the HMM, including the number of states, the transition format, the covariance matrix type, etc. In our example, a directory "hmms\proto" will be automatically created, and a prototype HMM will be generated inside this folder, as specified by SrcDir_init.
Conf_proto: this is the configuration file to generate the prototype HMM. You only need to specify the file path here. In our example, the file name is "puser.pcf," and it will be placed in "toolconfs" folder. This file will be automatically generated. You need to copy "makeProto.m" file from "files needed" folder to "exp" folder first.
Conf_init: this is the configuration file to initialize the models. You need to copy the file "hcompv.conf" from folder "files needed" into "toolconfs" folder, and specify the path here.

The output of the initialization step is a set of HMM models stored in the target folder.
TgtDir_init: this is the target folder for the initialization step. A set of new HMM models will be saved in the directory specified here. This directory is automatically generated. It is set to hmms\hmm1 in our example. Note that this set of models do not have "sp."

At this point, a set of models have been initialized.

The next step is embedded training. There are four global settings for this step.
Embed_train: "y" turns on embedded training; "n" turns it off
fix_sil: "y" means to introduce a "sp" model after the first low order model (does not include "sp") has been trained. Specifically, the "sp" model is copied from the central state of the silence model, and is a one state model (there are two non-emitting states, one emitting state). There is a direct transition between the entering and exit states, namely, the two non-emitting states, because there are actually no short pauses between words sometimes. The silence model is also modified. A forward and a backward transitions are added between the first and the third emitting states. You can refer to HTKbook section 3.2.2 for more details.

The code of the tool works in this way: before embedded training starts, it will first check whether a "sp" model is already in the model set loaded (specified by SrcDir_Embd). This is because sometimes, you have already trained a set of models including "sp," and you only want to modify this set of models, such as adding mixtures, or adding training iterations. In this case, you can simply turn off the initialization step, and only perform the embedded training step. If a "sp" model is found in the model set, "fix_sil" will be forced to "n," because there is no need to re-introduce the "sp" model and modify the silence model.

After the "sp" model is introduced, a couple of iterations of training will be performed according to "fix_iter" specification, then, the model mixtures will be split, starting from the second lowest mixture, according to the "numMixture" sequence.
fix_iter: this is how many training iterations after the "sp" model has been introduced.
embdOptStr: this is a pruning threshold for embedded training. Normally, you can set it to "-t 250.0 150.0 1000.0" and no need to change. Do not omit the " " on both sides. You can refer to HTKbook section 17.7 for details.

Seven input files/folders are needed as the input of embedded training.
SrcDir_embd: this is the source directory to load HMMs for embedded training. In our example, we have initialized a set of models. So, we need to load models from hmms\hmm1, where the initial models are stored.

You do not have to always load models from the initialization step. You can actually load models from anywhere. For example, if you already have a set of models in the "Final_hmmfolder," and you want to perform embedded training for this set of models, such as adding more mixtures, or adding more iterations for the models, you can set "SrcDir_embd" to "Final_hmmfolder."

hmmList_nosp & hmmList_sp: these are the HMM list with and without "sp." They are generated in the "transcription preparation" step. As stated, before embedded training begins, the tool will first check if a "sp" model already exists; if yes, "fix_sil" will be forced off, and "hmmList_sp" will be used; if no, the first low order model will be trained using "hmmList_nosp," and then, the tool will check "fix_sil;" if "fix_sil" is yes, then, the "sp" model will be introduced, and "hmmList_sp" will be used from then; if "fix_sil" is no, then, "hmmList_nosp" will be used all the way till the end.
monoMLF_nosp & monoMLF_sp : these are the phone level transcription with and without "sp." They are also generated in the "transcription preparation" step. Their usage is the same as that of hmmList_nosp & hmmList_sp.
Conf_embd: this is the configuration file for embedded training. You need to copy "herest.conf" from "files needed" folder to "toolconfs" folder, and specify the path here.
Conf_sil: this is the configuration file for fixing the silence. You need to copy "sil.hed" from "files needed" to "toolconfs" folder, and specify the path here. If you open "sil.hed" file, you will see a couple of commands such as "AT," "TI." "AT" means to add a transition between two states. "TI" means tie the designated states of two models

together, so that they will have the same mean and covariance matrix. You can refer to HTKbook section 17.8 for more details.

There are two outputs for embedded training.

TgtDir_embd: this is the target folder where the output HMMs are stored. In our example, the directory is hmms\hmm2.
Stat_embd: this is a statistic file generated by embedded training. This is only a byproduct. You won't need it in the forced alignment state.

At this point, the embedded training is done.

We have described how to train a set of models from the initialization step till the end. Yet, there are cases that you already have a set of models, and suppose you only want to create more mixtures based on this set of models. Suppose you have a set of 8 mixture models with "sp," and you want to create 16 mixture models. Here is how you do it with the tool:

a. turn off "Trans_prep" and turn on "Train_on," then turn off "Init."
b. turn on "Embed_train."
c. you can leave "fix_sil" either y or n. (set it to n if your 8 mixture models do not have "sp")
d. set SrcDir_embd to "hmms\fhmm" or to "hmms\hmm2," and set TgtDir_embd to "hmms\hmm2."
e. Set numMixture to 8;16. Do not set it to 16 only.
f. Set Iteration to 0;7. Do not set it to 7 only.

7.4. Tool_FA

After a set of initial HMMs have been trained, we come to the forced alignment step. Please copy "Tool_FA.m" and "Tool_FA.dcf" from "Tool_FA" folder to "exp." Then, open "Tool_FA.dcf" file.

In simple words, the forced alignment is an iterative processing. The phone transcriptions will be corrected using the set of models obtained in the training stage, then, the new transcription will be used to refine the models. This processing will be repeated for a couple of iterations. The final transcription will be the "perfect" version.

Again, at the beginning, there are three global options. "Trace_on" has the same meaning as in other tools.
Clean_up: "y" means to clean up the refined model before forced alignment begins, and also clean up the old aligned transcriptions before forced alignment.
LogDir: a progress report will be generated in this folder. The file name is "progress_FA.log." The report keeps track of how many iterations there are, and how much change was made in the aligned transcription after each iteration, etc.

After global settings, there are other settings for forced alignment.
FA_on: "y" turns on forced alignment; "n" turns it off.
FA_iteration: how many iterations are needed. One iteration includes: first, update the phone transcriptions using current models; second: re-train the models using the updated phone transcriptions.
Embd_iteration: this is how many embedded training iterations are needed to retrain the models each time after the transcriptions are updated. Again, the numbers are separated by ";" and the length of this sequence must be equal to FA_iteration. A zero means that the models will not be retrained after forced alignment.
Output_level: this specifies what you want to see in the aligned phone level transcriptions. "-o SWT" means that you only want the aligned phones. Scores (S), words (W) and time boundaries (T) will be suppressed. Do not forget the " " on both sides. You can refer to HTKbook section 17.8 for more details.
Prune_FA: this is a prune factor for the decoder in forced alignment mode. The smaller the first number is, the faster the decoder will be, but more sentences will not be successfully aligned. The bigger the last number is, the more sentences will survive, but the speed will also be slower. You can refer to HTKbook section 17.8 for more details.
Prune_embd: this is the same prune factor for embedded training as that in embdOptStr in Too_trainFA.dcf.
Init_phoneMLF: specify the initial phone level transcription file path (before any forced alignment iterations). Actually, the forced alignment processing itself does not need it. This is for statistical purpose. Each time after the transcription is updated, the tool will compute how much change between the new transcription and the old one, as you will see in the report. The first updated transcription will be compared with this initial one.

There are seven input files/folders needed by forced alignment and model refinement.
WordMLF: this is the word level MLF file generated in the "transcription preparation" step in Tool_trainFA.dcf.
Feat_List: this is the list of all the feature files to be aligned. This list is generated in the feature extraction step.
Dict: this is the dictionary file. Note that a "SENT_Boundary" should be in the dictionary. In our example, the dictionary has a "sp" after each word. If your model set does not include "sp," you should delete "sp" from this dictionary as well.
hmmList: a list of all the HMMs generated in the "transcription preparation" step in Tool_trainFA.dcf. This list needs to match the model set (whether there is a "sp" or not)
SrcDir_hmm: this is the folder to load the initial HMMs. In our example, all the initial HMMs are stored in "hmms\fhmm" folder. These HMMs will be used to do the first round of forced alignment.
Conf_FA: this is the configuration file for the decoder. Copy "hvite.conf" from "files needed" folder to "toolconfs" folder, and specify the file path here.
Conf_embd: this is the configuration file for embedded training.

There are three output files/folders generated.

aligned_folder: this is the folder to store the aligned transcriptions in each iteration. If Clean_up is turned on, the old aligned transcriptions will be cleaned up first. A series of

transcription files named as aligned_1.mlf, aligned_2.mlf,... will be generated. In our example, we specify this folder to be "labs."
TgtDir_hmm: this is the folder to store the retrained models in each iterations. In our example, a "refined_hmm" folder will be generated inside hmms folder, as specified. The refined models are used to do forced alignment in the next iteration.
Stat_embd: this is a statistic file generated by embedded training.

At this point, a set of aligned MLF files have been generated. The final "aligned_N.mlf" (in our example, N=6) is the "perfect" phone transcription. In addition, a set of well trained HMM models are stored in the folder "hmms\refined_hmm."

Since you have got a set of well trained HMMs, they can be used to align new data. Here are the main steps in aligning new transcriptions using a well trained model set.

a. Use the feature extraction tool (Tool_Compute_Feat.dcf) to extract features. Note that the feature type needs to match those used to train the models; for example, they are both 39 MFCC features.
b. In the tool for training (Tool_trainFA.dcf), turn on "Trans_prep," and "Gen_Word_MLF," turn off "Gen_Phn_MLF." So, the word level transcription of the new data will be generated. Turn off "Train_on."
c. Set FA_on to "y" in Tool_FA.dcf.
d. Set FA_iteration to 1. Since the models are well trained, you normally do not need multiple iterations. Of course, you can do that if you want.
e. Set Embd_iteration to 0. Again, since the models are well trained, you do not need to retrain them.
f. Set SrcDir_hmm to "hmms\refined_hmm," where the well trained HMMs are stored. Also, specify the list of HMMs in hmmList, in accordance to the refined HMMs.
g. Set "Init_phoneMLF" to "labs\aligned_1.mlf." Since there is actually no initial phone transcription for the new data (because the models exist already, so no need to generate initial phone transcription to train any models), yet a place holder is still needed, we can put the aligned MLF here. Of course, in the progress report, it will tell you no change was made between the "initial transcription" and aligned.mlf, since they are the same files. So, you can ignore the progress report in this case.

Now, you are all done with forced alignment. The final aligned transcription will be used in subsequent tools.

A working example is provided. The main output of the forced alignment tool package is put in the folder “files generated.” The database used is 863 Mandarin Chinese database. The phonetic transcriptions of 74 women speakers, namely, 34616 sentences, were forced aligned. The wave files, as well as the feature files are not included, since they are too big. The tool setup files used are exactly the same as the ones provided in the Tool_FA folder, as described in this manual. If you go through the steps in this manual using the same data, and same files as provided, you will get identical outputs. For feature extraction, 42 features were generated using the tfrontm MFCC method. These features consist of 12 DCTCs, 1 log energy, 1 pitch (14 static features), and delta, and delta-delta

terms. For training, 174 tonal phones were trained, including 27 Initials, 145 tonal Finals, and two silence models (sil and sp).

First, open “Log” folder. Inside this folder, you can see the progress reports for feature extraction (progress_feat.log), training initial models (progress_trainFA.log), and forced alignment (progress_FA.log). In progress_feat.log, it records that the frontend selected was “User,” and the feature dimension was 42. In progress_trainFA.log, you can see what happened in order. First, the tool did data preparation. This converted word transcriptions to word MLF, and then converted this word MLF to phone MLF. Then, the training started. The mixture splitting sequence as well as the iterations for each mixture splitting was recorded. The flat start initialization was performed first, and the embedded training followed. At the beginning of the embedded training, there was no “sp” model. After the first low order model (1 mixture) had been trained, the “sp” model was introduced, and the silence model was fixed. Then, a list of current number of mixtures being trained was given. After the desired order was achieved (16 mixtures), the models were copied to “hmms\fhmm” folder from “hmms\hmm2” folder. Finally, in progress_FA.log, you can see the progress of the forced alignment and model refinement. As you see, the forced alignment iteration was specified to be 6, and each time the phonetic transcriptions were updated, the models were retrained 3 times. In the last iteration, only forced alignment was performed, and the models were not refined (as a 0 was specified for the last iteration). After the forced alignment began, the tool checked how much change was made between the updated phonetic transcription and the old transcription obtained in the last round. For example, after the first iteration, it shows that the accuracy was 98.74%. This means that 1.26% of all the phones in the initial transcriptions (before forced alignment) were corrected by the first round forced alignment. Next, the models were retrained using this updated transcription. Then, the forced alignment was performed again, using the refined models. This time, the accuracy became 99.64%. This means that 0.36% of all the phones in the old transcriptions (obtained in the first round) were corrected by the second round forced alignment. This processing kept going until the desired number of iterations was achieved. Note that as the models kept being refined by the updated transcriptions, the accuracy between two consecutive transcriptions got higher. This shows that fewer and fewer phonetic pronunciations were updated by forced alignment, as most of them were already corrected by previous iterations.

Next, open “labs” folder. You can see 6 phone level MLF files generated by the forced alignment iterations. They were named as aligned_1.mlf, aligned_2.mlf, etc. In our example, aligned_6.mlf is assumed to be the “perfect” version, since it was generated by the last iteration. This version will be used in other parts of the ASR system. The files phone_nosp.mlf, and phone_sp.mlf are the initial phone transcriptions without and with “sp.” The file word.mlf is the word level MLF file of all the sentences.

Finally, open “hmms” folder. “fhmm” folder stores the HMM models before model refinement by forced alignment step. In other words, these models were trained by the initial phone transcriptions. In order to get these models, the initialization and embedded training were conducted. The models after initialization step were stored in “hmm1,” and

the models after embedded training were stored in “hmm2.” The “refined_hmm” folder stores the refined models. This set of refined models are well trained. It can be used to forced align new data as described by steps a-g above. The “proto” folder stores the model prototype. This prototype defines the model topology. In either fhmm, hmm1, hmm2, or refined_hmm folders, you can find two files, “hmmdefs” and “macros.” “hmmdefs” has all the parameters of all the models. “macros” defines some “global” parameters which are identical for all models, such as the feature vector dimension, and a variance floor. If any variance of any state in a model falls below this floor, it will be clamped to this floor. “macros” was generated in the initialization step and remained unchanged in all subsequent steps. “hmmdefs” got updated after each embedded training iteration.

8. A TOOL FOR SPEECH FEATURE EXTRACTION – TFRONTM GUIDE

8.1. Fundamentals of Speech Feature Extraction

ASR (Automatic Speech Recognition) is an automatic system which aims to convert voice to text. For processing the speech some steps should get considered. These steps are summarized as: collecting speech, preprocessing, feature extraction, recognition using models and outputting the text. Feature extraction is the focus of this section of the report.

The accuracy of recognition is highly depends on feature extraction so it is so important to compute “good” features. One of the popular feature types are MFCCs. The MFFC method is covered in elsewhere [1] , so the MFCCs are not discussed further here. The process of this feature extraction is summarized in figure 1.

In [3], a new feature extraction method is developed, with features referred to as DCTCs/DCSCs. The idea is to capture spectro-temporal patterns of the speech by using a discrete cosine transform (DCT) to compute DCTC/DCSC terms. DCTCs represent the spectral pattern in the frequency domain, and DCSCs represent the temporal pattern as DCTCs change over time. This method is also clearly described in [5].

8.2. Program Setup

1) Copy all the desired wave files to a wave folder. The default assigned folder in the code is “..\Data\train” for train data and “..\Data\train” for test data.
2) Make the wave file list for future use in “lists\trnw_exsa.lst” and “lists\tstw_exsa.lst”. There is a code provided called “`make_wave_dir.m`” which makes this list automatically. It reads the wave files from the default audio folder and copies the list to the default folder.
3) Define two setup files “Tool_ComputeFeat_train.dcf” and “Tool_ComputeFeat_test.dcf” which indicate the audio folder and extracted feature folders and files.
4) Feature extraction configuration file. One sample of this file is “`cp_dcs.ini`”. An explanation of how to make this configuration file is provided in section 6.

5) Create another configuration file called "`snr_801.trn`" (See section 4 for details).

8.3. Tool_ComputeFeat

8.3.1. Function

This Matlab function is designed to compute features and store them in feature files. There is an input for this function which contains some preliminary specifications. Some functions which are called within this function are as below:
HCopy
readhtk
tfrontm
Hcopy is a HTK software tool (© COPYRIGHT 2001-2009 Cambridge University Engineering Department) which is used to extract the MFCC features of the speech signal. For more detail refer to [1].
Readhtk, is an htk related file (designed by Mike Brookes [2]) which reads the htk parameter files.
Tfrontm, is a software generated by Binghamton University Speech lab group for DCTC/DCSC feature extraction [3]. For complete description please refer to section 4.

8.3.2. Use

The command line of Tool_ComputeFeat is as below:
`Tool_ComputeFeat(CmdFile)`
This CmdFile is a configuration file. In default name of this file is "Tool_ComputeFeat_train.dcf." The terms of this configuration files are as below:

Tool_ComputeFeat_train.dcf		
Terms	Default Value	Description
`Trace_on`	`Y`	Enables progress report for each tool if it is chosen to be "n", it only shows the current command line.
`Clean_up`	`Y`	Enables cleaning the target folder in each step.
`LogDir`	`Log`	This is the folder to store the progress log file. A "progress.log" file will be generated in this folder.
`Feat_On`	`y`	Enables the feature extraction.

Tool_ComputeFeat_train.dcf		
Terms	Default Value	Description
FrtEnd_opt	User	Some Frontend selections: HTK_MFCC, For MFCC frontend in HTK HTK_PLP, For PLP frontend in HTK User, User designed Frontend The default value is HTK_MFCC
Wave_List	lists\trnw_exsa.lst	The list of the wave files are shown in this file.
Feat_folder	..\Data\train	This is the folder which the extracted features are stored in.
Feat_List	lists\trn_exsa.lst	The list of the feature files.
Conf_MFCC	toolconfs\hcopy_MFCC.conf	This is the configuration file for MFCC feature extraction.
Conf_PLP	toolconfs\hcopy_PLP.conf	This is the configuration file for PLP feature extraction.
Conf_tfrontm	tfront\tfrontm.dat	This a configuration file for the tfrontm function.

8.4. Tfrontm

8.4.1. Function

This is the function which reads some input files, compute features, and writes them to an output file (*.mfc) using the wr_feat function. There are several other functions used by tfrontm. The key configuration files are tfrontm.dat (for some general specifications of tfrontm) and CP_FEA13.ini (for more detailed specifications).
The functions which are used directly by tfrontm are:
Cp_feat
wr_feat
Cp_feat.m does some calculations and computes the features (refer to section 5). Wr_feat writes the computed features in a specific order into desired (*.mfc) files.

8.4.2. Use

The command line for this function is:

`tfrontm(CmdFile,Wave_List,Feat_List,Feat_folder)`
where:
`CmdFile:` is generally names `tfrontm.dat` and contains some general properties of tfrontm function. More details are given in table below.
`Wave_List:` is the list of the wave files which are to be processed.
`Feat_List:` is the list of the feature files.
`Feat_folder:` is the folder of the feature files.

Tfrontm.dat		
Terms	Default Value	Description
`FILE_ID`	`TFRONT_SPEC`	This is File _ID
`SNR`	`300`	The SNR value
`FEAT_FILE`	`tfront\CP_FEA13.INI`	Default file name of the tfrontm function configuration. (See next table)
`FILE_TYPE`	`HTK`	Feature file formats: `TYPEA1, TYPEB1, or HTK`
`PARMTYPE`	`USER`	Parameter type of HTK

8.5. CP_feat

8.5.1. Function

This function has two modes, `init` and `proc`. This function does some calculations on wave files and returns the features (matrix) plus some specifications of the wave file.

8.5.2. Use

The command form is:
`[Feat,OutPars,addPars]=cp_feat(DoWhat,X,specFile, InitPars)`
The input parameters are:
`DoWhat`: indicates what type of function is going to be done. It can be either 'init' (which indicates doing some preliminary calculations, such as DCTC and DCSC basis vectors) or 'proc' (which computes the actual features).
`X:` samples read from a speech file (i.e., a speech waveform). This is only needed for the 'proc' mode.
`specFile:` is the configuration file (by default it is CP_fea13.ini.
`InitPars:` is a vector whose first element is the sample rate of the wave file and whose second element is the length of the Data (X) which is read from the audio file.
The outputs of cp_feat are:
`Feat:` Features (a matrix) which are extracted in 'proc' mode. In 'init' mode, Feat has no meaning.
`OutPars:` In 'init' mode, it returns a vector with 3 elements--Sample rate, Frame jump and Block jump.

addPars: In ‘init’ mode, it this is a vector with 12 character values. These values are number of DCTCs, number of DCSCs, frame length, Gammatone width, Window length of Δ, $\Delta - \Delta$, $\Delta - \Delta - \Delta$, Static warping type, Pitch tracker type and amplitude scaling.

8.5.2.1. INIT Mode

In this method first the content of the configuration file (CP_fea13.ini) is read by rd_spec function (see section 6) and the results are stored in the CP_Parse variable. Then a filter is initialized based on some defined poles and zeros using PreFilt function. Then filter bank weights are computed using genfw function (W). Based on the number of DCTCs and the frequency warping method, the frequency basis vector (bvF) is computed using genbv function.

If the dynamic type is selected as DCS, time basis vectors are also computed. This is done by again using the genbv function which incorporates time warping.
If DELTA is selected as the dynamic type, the genbv function is called with ‘d’ (which is defined within the function) and the length of the Δ.

8.5.2.2. PROC Mode

After computation of basis vectors, it is time to compute the features. The first step is to filter the speech data to with the PreFilt function. If pitch tracking type, which is defined using the Tracker_type option in CP_FEA13.ini, is set to anything but NONE (see section 6), a function named pitch_tracker calculates the pitch values. Based on the type of the pitch normalization chosen in CP_FEA13.ini file, the mean, variance or both of them are computed for the fro all pitch values for one file.

The next step is to compute the feature based on the audio file data. This data is extracted for each frame based on the Frame time and space and framed by a Kaiser window. The FFT of the signal is computed and the magnitude of this FFT is computed between 0 and Fs/2. If the Amplitude_scaling is set to LOG, the log of this magnitude is computed. Otherwise, using the number which is specified in this variable, the magnitude raised to the power of the number, is computed.

For energy computations, if the Log_Energy_Flag is enabled, the energy of of each frame of the signal is computed. If Log_Expand_Flag is enabled, the energy data is added to the end of features, otherwise the energy data will substitute the last line of computed features.

After computing the static features, dynamic features are computed. First the number of blocks to process are defined by dividing the Number of frames by block jump. Then the obtained feature blocks are multiplied to time basis vectors of ‘init’ mode.

8.6. Rd_spec

8.6.1. Function

This function is mainly designed to read the configuration file of the feature extraction and return the value to other functions.
This function is called in cp_feat function in ‘init’ mode.

8.6.2. Use

This function is used as below:

```
[Params,Use_term,LogEnergy,Param_char]=rd_spec(FileName,InitPars)
```

The inputs are:

`FileName:` The configuration file (default: CPFEA13.INI)

`InitPars:` It was described on section 5 as InitPars

Out puts:

`Params:` The numeric values of configuration file

`Use_term:` Number of used features

`LogEnergy:` A vector of some log energy flags

`Param_char:` The characteristic values of configuration file

As it was said on section 7.5, there is another configuration file which contains some more details about the feature extraction. This file is defined in main root and is copied to tfront folder in the beginning of running the code. These file specifications are as below:

CP_Fea13.ini (configuration file)		
Variable	Default Value	Description
Basic parameters		
Sample_rate	16000 Hz	Sample rate
Frame_length	8 ms	
Frame_space	2 ms	
FFT_length	256 points	
Kaiser_window_beta	6	
Prefilt_Center_Freq	3200 Hz	Pre-filtering center (0 means no pre-filter)
Spectral_range	40 dB	
Spectral Analysis		
Low_freq_limit	100 Hz	0~300 Hz
High_freq_limit	7000 Hz	3000~8000 Hz
Amplitude_scaling	LOG	It can be LOG or any value which will be the power.
Numb_filters	26	Number of filters in filterbank
plot_spec	0	Enables the plot of spectrogram
CP_Fea13.ini (configuration file)		
Variable	Default Value	Description
shift_deg_NonSym	0.5	Shifting degree using non-symmetric window
Log_Energy_Flag	1	Enables adding energy as one of the features
Log_Expand_Flag	0	Enables adding the energy feature as an additional feature or substitutes the latest feature.
Width_gammatone	1.0	Width of Gammatone filter
Static features		
Numb_dctcs	13	
DCTC_type	FFT	FFT, MEL , GAMMA (Gammatone)
Static_warp_type	MEL	NONE, MEL, BILINEAR
Static_Warp_factor	0.15	
Dynamic features		
Dyn_Type	DCS	DCS, DELTA
Numb_dyn_terms	3	Number of dynamic terms. It is inactive if DELTA is chosen for Dynamic type.
Time_warp_type	KAISER	KAISER, GAUSSI, SIGMOI, NONSYM
Time_warp_factor	25	
Block_length	151	
Block_jump	4	
BVF_norm_flag	0	
BVT_norm_flag	0	
Delta_window_length	2	Window length of Δ
Accelator_window_length	2	Window length of Δ-Δ
Delta3_window_length	0	Window length of Δ-Δ-Δ
Pitch Features		
Tracker_type	NONE	NONE, YAAPT, YIN, PRAAT
All_part_voice_yappt	0	Used to control all voiced (0) or partially voiced (1) in YAPPT
Pitch_Normalization	NONE	NONE, MEAN, VARIANCE, BOTH
Use_term*	NONE	USER, NONE (NONE uses all features)

* For Use_term, there is a table at the end of the file. If this variable is set to NONE, this table will be skipped, otherwise all the DCTCs and related DCSCs are used. Use_term is basically used to eliminate some of the computed DCSC/DCSC terms from the feature matrix.

9. REFERENCES

[1] R.E. Bellman, *Dynamic Programming*, Dover Publications Reprint edition, March 4, 2003.

[2] G.E. Peterson and H.L. Barney, "Control methods used in a study of the vowels," *J. Acoust. Soc. Am.*, vol.24, no.2, pp.175-184, March. 1952.

[3] H. Hermansky, "Perceptual linear prediction analysis of speech," *J. Acoust. Soc. Am.*, vol.87, no.4, pp.1738-1752, Apr,1990.

[4] B.P. Bogert, M.J.R. Healy and J.W. Tukey, "The quefrency analysis of time series for echoes: cepstrum, pseudo autocovariance, cross-cepstrum and Saphe cracking," *Proceedings of the Symposium on Time Series Analysis* (M. Rosenblatt, Ed) Chapter 15, pp.209-243, New York: Wiley, 1963.

[5] E. Zwicker and H. Fastl, *Psychoacoustics, Facts and Models*, Chapter 3, pp.25-28, Springer-Verlag 1990.

[6] J.S. Bridle and M.D. Brown, "An experimental automatic word-recognition system," *JSRU Report*, no.1003, Joint Speech Research Unit, Ruislip, England, 1974.

[7] S.S. Stevens, J. Volkmann and E.B. Newman, "A scale for the measurement of the psychological magnitude pitch," *J. Acoust. Soc. Am.*, vol.8, no.3, pp.185-190, 1937.

[8] X. Zhang, M.G. Heinz, I.C. Bruce, and L.H. Carney, "A phenomenological model for the response of auditory-nerve fibers: I. nonlinear tuning with compression and suppression," *J.Acoust. Soc. Am.*, vol.109, no.2, pp.648-670, Feb.2001.

[9] H. Fletcher, "Auditory patterns," *Reviews of Modern Physics*, vol.12, Jan. 1940.

[10] D.W. Robinson and R.S. Dadson, "A redetermination of the equal-loudness relations for pure tones**,**" *British Journal of Applied Physiology***,** vol. 7, pp.166-181, 1956.

[11] J. Makhoul, "Linear prediction: a tutorial review," *Proceedings of the IEEE*, vol.63, pp.561-580, 1975.

[12] L. Rabiner and R. Schafer, *Digital Processing of Speech Signals*, Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1978.

[13] P.D. Patterson, K. Robinson, J. Holdsworth, D. McKeown, C. Zhang, and M.H. Allerhand, "Complex sounds and auditory images," in *Auditory and Perception*. Oxford, UK: Y. Cazals, L. Demany, and K. Horner, (Eds), Pergamon Press, 1992, pp.429–446.

[14] M. Slaney, "An efficient implementation of the Patterson-Holdsworth auditory filter bank," *Apple Technical Report,* no.35, Advanced Technology Group, Apple Computer, Inc., Cupertino, CA, 1993.

[15] S. Memon, M. Lech and N. Maddage, "Speaker verification based on different vector quantization techniques with Gaussian mixture models," in *Third Int. Conf. on Network and System Security*, 2009, pp.403-408.

[16] H.S. Jayanna and S.R.M. Prasanna, "Fuzzy vector quantization for speaker recognition under limited data conditions," *TENCON 2008-IEEE Region 10 Conference*, 2008, pp.1-4.

[17] J. Chen, K.K. Paliwal, M. Mizumachi and S. Nakamura, "Robust MFCCs derived from different power Spectrum," in *Eurospeech 2001,* Scandinavia, 2001.

[18] C. Wang, Z. Miao and X. Meng, "Differential MFCC and vector quantization used for real-time speaker recognition system," in *IEEE Congress on Image and Signal Processing,* 2008, pp.319-323.

[19] R. Drullman, J.M. Festen and R. Plomp, "Effect of reducing slow temporal modulations on speech reception," *J. Acoust. Soc. Am.*, vol. 95(5), pp.2670-2680, 1994.
[20] M. Athineos, H. Hermansky and D.P.W Ellis, "LPTRAPS: linear predictive temporal patterns," in *Proc. of Interspeech*, Jeju Island, Korea, pp. 1154-1157, 2004.
[21] F. Valente and H. Hermansky, "Hierarchical and parallel processing of modulation spectrum for ASR applications," in *ICASSP-2008,* April 2008, pp.4165-4168.
[22] M. Kleinschmidt, "Methods for capturing spectro-temporal modulations in automatic speech recognition," *Acustica united with acta acustica*, vol. 88, pp. 416–422, 2002.
[23] M. Kleinshmidt, "Localized Spectro-Temporal Features for Automatic Speech Recognition," in *Eurospeech 2003,* Sept. 2003, Switzerland.
[24] T. Gramß, "Fast algorithms to find invariant features for a word recognizing neural net," in *IEEE 2nd International Conference on Artificial Neural Networks*, Bournemouth, 1991, pp. 180–184.
[25] J. Allen, "Short term spectral analysis, synthesis, and modification by Discrete Fourier Transform," *IEEE Trans. Acoust., Speech, and Signal Processing*, vol. ASSP-25, no. 3, pp. 235-238, June. 1977.
[26] C. Kim and R.M. Stern, "Feature extraction for robust speech recognition using a power-law nonlinearity and power-bias subtraction," in *INTERSPEECH-2009*, Sept.2009, pp.28-31.
[27] D. O'Shaughnessy, *Speech communication: human and machine*. Addison-Wesley, 1987. pp.150.
[28] J.O. Smith and J.S. Abel, "The Bark bilinear transform," in *Proceedings of the IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, New Paltz, New York, IEEE Press, Oct. 1995.
[29] S. Wang, A. Sekey and A. Gersho, "An objective measure for predicting subjective quality of speech coders," *IEEE Journal on Selected Areas in Communications*, vol.10, no.5, pp.819-829, June 1992.
[30] H. Duifhuis, "Consequences of peripheral filter selectivity for nonsimultaneous masking," *J. Acoust. Soc. Am.*, vol.54, no.6, pp.1471-1488, 1973.
[31] G.M. Bidelman and A.S. Khaja, "Spectrotemporal resolution tradeoff in auditory processing as revealed by human auditory brainstem responses and psychophysical indices," *Neuroscience Letters*,vol. 572, pp. 53-57, 2014.
[32] M.J. Shailer and B.C.J. Moore, "Gap detection as a function of frequency, bandwidth, and level," *J. Acoust. Soc. Am.*, vol.74, no.2, pp.467-473, Aug. 1983.
[33] B.Meyer, S.V. Ravuri, M.R. Schadler and N. Morgan, "Comparing different flavors of spectro-temporal features for ASR," in *INTERSPEECH-2011*, Aug.2011, pp.1269-1272.
[34] D.A. Depireux, J.Z. Simon, D.J. Klein, and S.A. Shamma, "Spectro-temporal response field characterization with dynamic ripples in ferret primary auditory cortex," *J. Neurophysiol.*,vol.85, pp.1220–1234, 2001.
[35] W. Ge, "Two modified methods of feature extraction for automatic speech recognition," Master thesis, Department of Electrical and Computer Engineering, Binghamton University, Dec.2013.
[36] H. Hermansky and N. Morgan, "RASTA processing of speech," *IEEE Trans. Speech and Audio Processing*, vol.2, no.4, pp. 578–589, 1994.

[37] H. Hermansky and S. Sharma, "TRAPS - Classifiers of temporal patterns," in *ICSLP*, 1998, vol.3, pp. 1003–1006.
[38] X. Liu and S.A. Zahorian, "A unified framework for filterbank and time-frequency basis vectors in ASR frontends," to appear in *ICASSP-2015*, Brisbane, Australia, Apr. 2015.
[39] B.Y. Chiu, R. Bhiksha and R.M. Stern, "Towards fusion of feature extraction and acoustic model training: a top down process for speech recognition," in *INTERSPEECH-2009*, Sept.2009, pp.32-35.
[40] B.Y. Chiu and R.M. Stern, "Analysis of physiologically-motivated signal processing for robust speech recognition," in *INTERSPEECH-2008*, Sept.2008, pp.1000-1003.
[41] V. Zue, S. Seneff, and J. Glass, "Speech database development at MIT: TIMIT and beyond," *Speech Communication*, vol. 9, pp. 351-356, 1990.
[42] S. Young et al., "The HTK Book (for HTK Version 3.4)," retrieved from http://htk.eng.cam.ac.uk/. Cambridge University, revised for HTK Version 3.4 in March 2009.
[43] K. Lee and H. Hon, "Speaker-independent phone recognition using Hidden Markov Models," *IEEE Trans. on Acoust., Speech, and Signal Processing*, vol.37, no. 11, pp.1642-1648, Nov.1989.
[44] Z.B. Nossair, P.L. Silsbee and S.A. Zahorian, "Signal modeling enhancement for automatic speech recognition," in *Proceedings of ICASSP-95*, vol. 1, pp.824-827, 1995.
[45] S.A. Zahorian and B. Wong, "Spectral amplitude nonlinearities for improved noise robustness of spectral features for use in automatic speech recognition," *J. Acoust. Soc. Am.*,, vol. 130, no.4, pp.2524, 2011.
[46] N. Mesgarani, G.S.V.S. Sivaram, S.K. Nemala, M. Elhilali and H. Hermansky, "Discriminant spectrotemporal features for phoneme recognition," in *INTERSPEECH-2009*, Sept.2009, pp.2983-2986.
[47] A. L. Ronzhin, R. M. Yusupov, I. V. Li, and A.B. Leontieva, "Survey of Russian Speech Recognition Systems," in *Proc. SPECOM '06*, St. Petersburg, Russia, pp. 54 – 60, June 2006.
[48] A. L. Ronzhin and A. A. Karpov, "Large Vocabulary Automatic Speech Recognition for Russian Language," *SPIIRAS*, St. Petersburg, Russia, 2006.
[49] A. A. Zaliznjak, *Grammatical Dictionary of the Russian Language*, 4th edition, Moscow, Russia, 2003.
[50] S. Zablotskiy, A. Shvets, M. Sidorov, E. Semenkin and W. Minker, "Speech and Language Resources for LVCSR of Russian," Institute of Communications Engineering, University of Ulm, Germany, 2012.
[51] E. Whittaker and P. Woodland, "Comparison of Language Modelling Techniques for Russian and English," in *Proc. ICSLP '98*, Sydney, Australia, 1998.
[52] E. Whittaker and P. Woodland, "Particle-based language modelling," in *Proc. ICSLP '00*, Beijing, China, Oct. 2003.
[53] Kuznetsova, A. I. and T. F. Efremova, *Dictionary of Morphemes of the Russian Language*, 1986.
[54] *Russian Grammar – Morpheme database*, Institute of Russian Language, RAS, http://rusgram.narod.ru/morf1t.html, 2014.
[55] A. N. Tikhonov, *Morphemic Spelling Dictionary*, Astrel, 2002, http://slovari.yandex.ru/.

[56] Decomposer, http://slovonline.ru/slovar_sostav/.
[57] D. Kanevsky, M. Monkowski, and J. Sedivy, “Large vocabulary speaker-independent continuous speech recognition in Russian language,” in *Proc. SPECOM ’96*, St. Petersburg, Russia, pp. 28 – 31, 1996.
[58] V. A. Barannikoff , and A. A. Kibkalo, “The software package for constructing speech recognition systems,” In *Proc. of the III All-Russian Conference "Theory and practice of speech research" APCO-2003, Moscow State University*, Moscow, Russia, pp. 7 – 12, 2003.
[59] A. A. Karpov and A. L. Ronzhin, “Speech Interface for Internet Service Yellow Pages,” in *Proc. International IIS: IIPWM'05*, Gdansk, Poland, pp. 219 – 228, June 13-16, 2005.
[60] S. Young et al., *The HTK Book*, Cambridge University, 2009.
[61] S. A. Zahorian et al., “Open Source Multi-Language Audio Database for Spoken Language Processing Applications,” in *Proc. INTERSPEECH*, 12th Annual Conference of the International Speech Communication Association, Florence, Italy, August 27-31, 2011.
[62] E. W. D. Whittaker, “Statistical Language Modelling for Automatic Speech Recognition of Russian and English,” PhD Dissertation, University of Cambridge, Cambridge, 2000.
[63] A. L. Ronzhin and A. A. Karpov, “Implementation of Morphemic Analysis for Russian Speech Recognition,” *in Proc. SPECOM ’04*, St. Petersburg, Russia, pp. 291-296, 2004.
[64] Z.-H. Tan and B. Lindberg, “Low-complexity variable frame rate analysis for speech recognition and voice activity detection,” IEEE J. of Selec. Top. in Sig. Proc., vol. 4, no. 5, pp. 798 – 807, 2010
[65] Milner, B., “Inclusion of Temporal Information into Features for Speech Recognition,” Proc. ICSLP 96, pp. 256-259, 1996.
[66] Qifeng Zhu and Abeer Alwan, “On the use of variable frame rate analysis in speech recognition,” ICASSP, pp. 3264–3267, 2000.
[67] K.M. Pointing and S.M. Peeling, “The use of variable frame rate analysis in speech recognition,” Computer Speech and Language, vol. 5, no. 2, pp. 169–179, April 1991.
[68] Philippe Le Cerf and Dirk Van Compernolle, “A new variable frame rate analysis method for speech recognition,” IEEE Signal Processing Letter, vol. 1, no. 12, pp. 185–187, December 1994.
[69] J. Macías-Guarasa, J. Ordónez, et al., “Revisiting scenarios and methods for variable frame rate analysis in automatic speech recognition,” Eurospeech, pp. 1809–1812, 2003.
[70] Zahorian, S. and Jagharghi, A. “Spectral-shape Features versus Formants as Acoustic Correlates for Vowels,” J. Acoust. Soc. Amer., vol. 94, pp 1966-1982, 1992.
[71] L. Rabiner and B. H. Juang, Fundamentals of speech recognition.: Prentice-Hall, Inc., 1993.
[72] S. B. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," IEEE Trans. on Acoustics, Speech and Signal Processing, vol. 28, pp. 357-366, 1980.
[73] H. You, Q. Zhu, and A. Alwan, “Entropy-based variable frame rate analysis of speech signals and its application to ASR”, in Proc. IEEE ICASSP, 2004.”

[74] Furui, S., “On the Use of Hierarchical Spectral dynamics in Speech Recognition,” Proc. ICASSP 90, pp. 789-792, 1990.
[75] Furui, S., “Speaker-Independent Isolated Word Recognition Using Dynamic Features of Speech Spectrum,” IEEE Trans. ASSP, vol. 34, pp. 52-59, February 1986.
[76] S. J. Young et al., HTK: Hidden Markov Model Toolkit V3.4, Reference Manual. Cambridge, U.K.: Cambridge Univ. Speech Group, 2004.
[77] S.J.Young, J.J.Odell,P.C.Woodland, "Tree-Based State Tying for High AccuracyAcoustic Modelling," Cambridge University, Engineering Department.

APPENDIX A – PUBLICATIONS AND PRESENTATIONS

A1 X. Liu and S.A. Zahorian, A Unified Framework for Filterbank and Time-Frequency Basis Vectors in ASR Frontends, accepted for publication in ICASSP 2015, April 2015

A UNIFIED FRAMEWORK FOR FILTERBANK AND TIME-FREQUENCY BASIS VECTORS IN ASR FRONTENDS

Xiaoyu Liu, Stephen A. Zahorian

Department of Electrical and Computer Engineering, Binghamton University, Binghamton, NY, 13902, USA

ABSTRACT

For many years, filterbanks have been widely used as one step of frontend feature extraction for Automatic Speech Recognition (ASR). In this paper, we propose a unified framework for ASR frontends, by first moving the nonlinear amplitude scaling, and then combining the filterbank weights with the cosine basis vectors. As part of this framework, we show that the delta terms used to encode feature dynamics can be viewed as one realization of a set of "unified" basis vectors over time. With this framework, frontends can be developed, analyzed and evaluated through their basis vectors over frequency and time.

Index Terms— Filterbank, spectro-temporal, unified, basis vector, frontend

1. INTRODUCTION

For many years, filterbanks, implemented as weighted sums of FFT magnitudes, are widely used as a frontend processing step for ASR systems. Figure 1(a) is a block diagram of the filterbank-based feature extraction approach. One commonly used version of this approach is to compute Mel Frequency Cepstral Coefficients (MFCCs) [1]. The MFCC features are computed using a set of triangular bandpass filters approximately logarithmically spaced above 1 kHz to map the short time power in the Hz domain to the Mel domain. In recent years, various enhanced MFCC algorithms have been developed. In [2], a SMFCC algorithm incorporates the pitch frequency information in building the filterbank, and in [3], the spectral envelope of the voiced frames is enhanced to improve the noise-robustness of the MFCCs.

To extract features from the amplitude-scaled output of the filterbank, the Discrete Cosine Transform (DCT) is computed using "half" cosine multiple basis vectors. The feature calculation using these "regular" cosine basis vectors is given by equation (1) as:

$$DCTC(i) = \sqrt{\frac{2}{N}} \sum_{j=1}^{N} a\big(P(j)\big) \cos\left(\frac{\pi i}{N}(j - 0.5)\right) \qquad (1)$$

where $DCTC(i)$ is the *i*th DCT coefficient, *N* the total number of filter channels, *P*(*j*) the output power of the *j*th channel, and *a*() the amplitude scaling function. The DCT coefficients are similar to the principal components of the spectrum. In [4], a Distributed DCT (DCT-II) method is presented to remove the correlation between filterbank outputs more completely, which leads to a more compact set of cepstral features.

As pointed out in [5,6,7,8], the delta and acceleration terms of the DCTCs greatly help to improve the system accuracy since these time derivatives capture the dynamic behavior of adjacent coefficients. The delta terms are computed through equation (2), where Θ is the window length in frames, and higher order terms are the deltas of lower order ones.

$$\Delta(t) = \frac{\sum_{\theta=1}^{\Theta} \theta (DCTC_{t+\theta} - DCTC_{t-\theta})}{2\sum_{\theta=1}^{\Theta} \theta^2} \qquad (2)$$

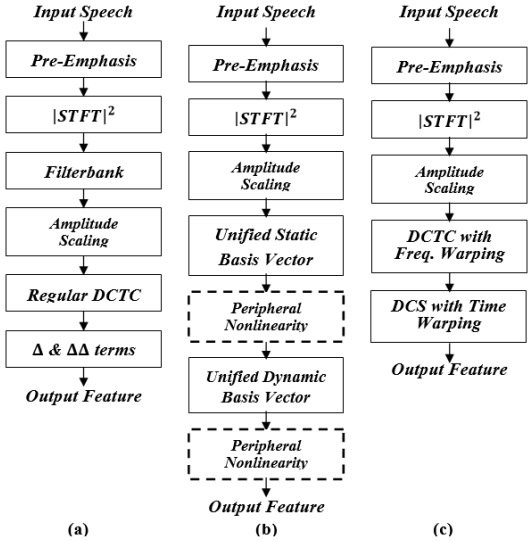

Fig.1. Block diagrams of the filterbank-type frontend (a), the unified structure (b), and the spectro-temporal system (c) in [9].

Spectro-temporal frontends capture much more detailed dynamic information of the spectrum than the time derivative terms. The work in [9] provides finer time resolution by weighting time blocks of the static features with a set of Discrete Cosine Series (DCS) expansion, and in [10], parallel and hierarchical structures are developed based on a temporal filterbank and in [11], two-dimensional Gabor-type features are obtained to describe the diagonal spectro-temporal patterns.

In our work, we propose a unified framework for ASR frontends, which is built upon a set of unified basis vectors over time and frequency. The nonlinear amplitude scaling is moved to immediately after the FFT magnitude step. Under this framework, frontend systems, such as (but not limited to) [9, 11], can be characterized entirely through the unified basis

vectors, which thus gives a common yardstick for analyzing frontends. We also discuss other potential benefits of this perspective.

2. A UNIFIED FRAMEWORK

2.1. Moving the amplitude scaling to the 'front'

It's interesting to note that if we move the nonlinear amplitude scaling in Figure 1(a) to before the filterbank, the filterbank weights can then be combined with the regular basis vectors by a simple matrix multiplication. However, this switching should be justified by inherent auditory properties as well as ASR experiments.

Physiologically, as the sound wave travels along the basilar membrane in the cochlea, different frequency components cause the amplitude of the basilar membrane vibration at different areas. Thus, auditory models, such as the Seneff model [12], use filterbanks to represent this frequency analysis property. The bandwidth is designed to match the ears resolution. The membrane vibration causes electrochemical transformation by the hair cells in corresponding areas of the membrane, thus "fires" the neurons by sending out spikes with a certain strength. The nonlinear amplitude scaling characterizes the neuron firing rate varying with different sound intensities. A commonly used nonlinearity is the logarithmic compression. However, indicated by more sophisticated auditory models such as [13], this neuron firing curve shows an "S" shape, thus, can be better approximated by a power-law function [14].

Based on this physiological scheme, it's not completely accurate to say whether the nonlinear scaling should be before or after the filterbank in a rigorous sense because the nonlinearity should also be frequency-dependent, since the hair cells at different areas of the basilar membrane shows distinct sensitivity to the sound intensity. Directly mapping the original spectrum with the nonlinearity inherently eliminates this distinction. However, we place the nonlinearity before the filterbank since (1) frequency-independency simplification is made and experimentally justified by ASR systems, such as MFCC and PLP [15], which uses an equal-loudness curve to compensate for the simplification, (2) based on (1), there is no compelling evidence as to where the nonlinearity should be placed, (3) experimentally, we will show that it does not matter much to move the nonlinearity to before the filterbanks, and (4), as discussed below it allows the system unification which brings benefits.

2.2. Unified basis vectors

First, with the amplitude scaling moved, it's easy to create a set of "unified" static basis vectors by a matrix product. Suppose the rows of the matrix $\boldsymbol{W}$ contain the filterbank channel response, and the rows of $\boldsymbol{BVF_{reg}}$ contain the regular static cosine basis vectors, the unified version $\boldsymbol{BVF_{uni}}$ is given in (3), and the amplitude-scaled FFT spectrum is weighted by $\boldsymbol{BVF_{uni}}$ to obtain the static DCTCs.

$$\boldsymbol{BVF_{uni}} = \boldsymbol{BVF_{reg}}\boldsymbol{W} \tag{3}$$

In the standard MFCC framework, the dynamic (Δ) features are computed from the static DCTCs, using Eq.2. Here we show that the Δ terms can also be computed using basis vector manipulations. From (2), to compute any *n*th order differential term, its basis vector with respect to the previous lower order (neglecting the constant denominator) is given by $\boldsymbol{bv}_n = [-\theta_n, -\theta_n + 1, \ldots, 0, 1, \ldots \theta_n]$, where θ_n is the window length. If we view $\boldsymbol{bv}_n$ as a discrete signal, with each element representing both the amplitude and the time index (i.e. [-2,-1,0,1,2] gives a signal whose magnitude is -2 at index -2, and -1 at index -1, etc.), then, the *n*th order basis vector with respect to the DCTCs can be computed as:

$$\boldsymbol{bvT_n} = \boldsymbol{bv_1} \circledast \boldsymbol{bv_2} \ldots \circledast \boldsymbol{bv}_n \tag{4}$$

where $\circledast$ is the convolution operator, and each $\boldsymbol{bv_i}$ is the *i*th order basis vector in terms of its previous lower order. Thus, putting all $\boldsymbol{bvT_n}$, including the zeroth order, into rows of a unified dynamic basis vector matrix $\boldsymbol{BVT_{uni}}$, the final feature matrix $\boldsymbol{F}$ at the output is given by (5), where $a(\boldsymbol{X})$ is the amplitude-scaled FFT spectrum.

$$\boldsymbol{F} = \boldsymbol{BVT_{uni}} \cdot [\boldsymbol{BVF_{uni}} \cdot a(\boldsymbol{X})]^T \tag{5}$$

2.3. Discussion

In this section, we present a detailed discussion on the significance/applications of this unified frontend perspective, whose block diagram is depicted in Figure 1(b).

First, it's important to note that $\boldsymbol{BVT_{uni}}$ and $\boldsymbol{BVF_{uni}}$ in (5) can take on any generalized forms, though they are derived from a specific category of frontends. On a higher level, Eq.5 shows that features can be viewed as a series of linear transformations of the spectrum scaled by an auditory nonlinearity, with optional peripheral nonlinearities in between (dashed blocks in the diagram). These linear transformations are represented by the unified basis vectors. Filterbanks (or other parts) exert their impact on system quality by shaping the basis vectors implicitly. Thus, the unified basis vectors hold and determine the properties of a frontend. In this sense, the scheme gives us a common "yardstick" to analyze and compare frontends which appear to be different or similar based on the properties of the unified basis vectors.

The first example to illustrate this point is the comparison between the "standard" MFCC and the spectro-temporal system in [9], whose diagram is given in Figure 1(c). It's important to emphasize that in the unified framework, both systems compute features in a mathematically identical manner, and the only difference lies in the unified basis vector forms. In [9], in computing the DCTCs, the *i*th basis vector $\phi_i(f)$ over frequency f is given by (6):

$$\phi_i(f) = \cos[\pi i \cdot g(f)] \cdot \frac{dg}{df} \tag{6}$$

where $g(f)$ is a frequency warping function. In Figure 2, we plot the first 3 basis vectors (left) with $g(f)$ set to a Mel-shape warping (right), and in Figure 3, we also plot the first 3 unified basis vectors in the MFCC using a 26-channel Mel filterbank.

The unified basis vectors produced by the Mel filterbank are less smooth than the ones generated by the Mel-shape warping. In Figure 2, the Mel scale is represented in a continuous manner; however, for the case in Figure 3, the basis vectors are computed using a 26 step quantized Mel scale.

Thus, we might expect finer frequency resolution for the Mel-shape warping approach, which might lead to better recognition accuracy. However, the difference should be small, since they are essentially two ways of implementing a Mel warping, which can be seen by the similarities of the basis vectors.

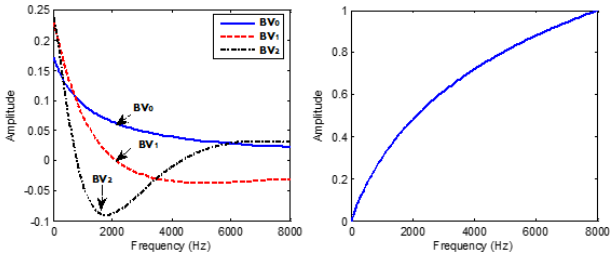

Fig2. First 3 DCTC basis vectors (left) with an embedded Mel-shape warping (right) in the system of [9].

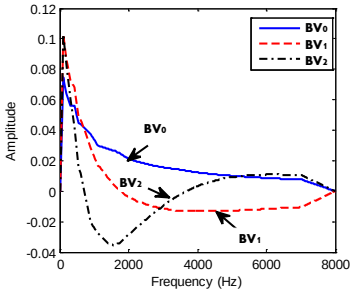

Fig.3. First 3 unified DCTC basis vectors for the standard MFCC frontend. A 26-channel Mel filterbank is combined with regular cosine basis vectors.

To obtain dynamic features, the system in [9] uses a set of Discrete Cosine Series (DCS) basis vectors to weight the time blocks of the DCTCs. The *i*th DCS basis vector is defined in (7):

$$\psi_i(t) = \cos[\pi i \cdot h(t)] \cdot \frac{dh}{dt} \qquad (7)$$

where $h(t)$ is a time warping function. Again, the first 3 DCS basis vectors are plotted in Figure 4 (left) with a continuous Kaiser window as the time warping, and in the right panel, the first 3 differential basis vectors are presented, with BV0 refers to the zeroth order in both cases.

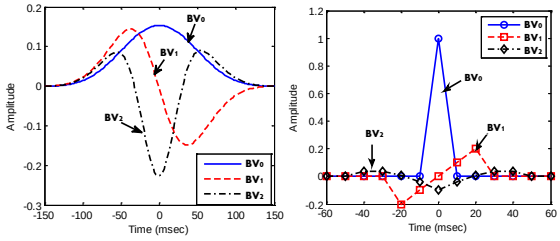

Fig.4. First 3 DCS (left) and differential (right) basis vectors in computing the dynamic features in system [9] and MFCC.

Clearly, the discrete differential (indicated by markers) and continuous DCS basis vectors are very different, both from their "look" and the logic used to derive them. However, as they are put into the same unified framework, we are able to analyze system properties through these basis vectors. If we compare the zeroth order, DCS from [9] puts more weight on the block center and gradually reduces on both sides, whereas the differential case uses only the block center term. We infer that DCS from [9] may provide finer time resolution by "magnifying" details on different sections of a block through its basis vectors.

Another system that can be analyzed in the unified framework is found in [11,16], where a set of Gabor filter based features are proposed to capture the Localized Spectro-Temporal Features (LSTFs). However, the work of [17] shows that LSTFs can be obtained through weighting a rotated spectrum by the basis vectors, and since rotation of the spectrum is equivalent to rotating the basis vectors, the LSTF frontends are covered by the unified framework. In [17], phonetic recognition results are obtained at different angles of the rotation, which can be visualized and thus analyzed by the rotated basis vectors.

Potentially superior features can also be developed through the unified concept. As one example, motivated by the time-frequency resolution insight of the unified basis vectors, the static and dynamic basis vector steps could be interchanged. This would allow the use of frequency-dependent dynamic basis vectors, with better time resolution for the higher frequency terms. Specifically, for a time-frequency block of the spectrum, first fix the frequency index, and weight the spectrum by a set of basis vectors over time, in which the time warping is a two-dimensional function that has broader shape at low frequencies than at high frequencies. Then, use another set of basis vectors over frequency to weight the output of the previous step, in which a frequency warping is embedded. As another example, motivated by the general forms of the unified basis vectors, we can use a non-symmetric time warping window that emphasizes more on the left section of the block than on the right. The auditory clue is in [18], that the left context of a phoneme is more informative than the right context. Two Gaussian windows with different variances can be combined to construct this warping.

However, it should be pointed out that there are limitations to this unified framework. It should not be viewed as a framework that replaces frontends, nor even accounts for all of them (e.g. PLP). However, for many cases, it reveals the essence of features with a straightforward tool, the unified basis vectors, as a linear transformation. Possibly more effective systems can be developed. For frontends which might not fully fall into this structure, their system properties can still be studied with the view presented here. Also, the filterbank and the regular basis vectors can still be implemented in two separate steps as needed, to allow various techniques, such as the PNCC algorithms [14,19] to be inserted.

3. EXPERIMENTAL EVALUATION

The goal of this section is to present the system performance purely in terms of the unified basis vectors built from various filterbanks and the system in [9]. Extensive tests were also conducted to determine the effects of moving the nonlinearity.

3.1. Phonetic level recognition task

A 39 phoneme recognition task with TIMIT was conducted. 3696 and 1344 utterances (SA sentences removed) were used for training and testing respectively. 48 3-state 32-Gaussian-

mixture monophone HMMs were trained by HTK 3.4, and a phonetic bigram language model was used for decoding. Throughout this subsection, the optimal frame length and space for differential dynamic basis vectors were 25ms and 10ms respectively, and for the example spectral-temporal system of [9] were 8ms and 2ms. The optimal block length and space for computing DCSCs were 302ms and 8ms (151 and 4 frames). 26 and 40 channels were used for Mel and gammatone filterbank derived basis vectors respectively. The gammatone was implemented as in [20].

In Table 1, we examine the effect of placing the amplitude scaling before the filterbank. Logarithmic and power functions were tested. For the Mel and gammatone cases, the static and dynamic basis vectors were 12 regular cosine as in Eq.1 (plus one log-energy term) and delta/acceleration (39 features in total), and the power was set to 0.1. To make more thorough tests of moving the nonlinearity, PLP frontends were also implemented, though the analysis of this frontend in the unified framework may not be straightforward. MATLAB code to obtain the PLP results can be found in [21], where 16 trapezoids were used as the filterbank with a Hynek's 'magic' equal-loudness curve built into the weights, and the power value was 0.33. 12 static terms were obtained from the LPC cepstral recursion. The dynamics were delta and acceleration. In the baseline cases (bolded), the amplitude scaling was placed after the filterbanks.

Table 20. Phonetic accuracy (%) of placing the amplitude scaling before/after the filterbanks

FB Type	Scaling Type	Scaling pos.	Accuracy
Mel	**log**	**After FB**	**69.8**
Mel	log	Before FB	69.6
Mel	**power (0.1)**	**After FB**	**69.8**
Mel	power (0.1)	Before FB	69.7
Gammatone	**log**	**After FB**	**70.3**
Gammatone	log	Before FB	70.1
Gammatone	**power (0.1)**	**After FB**	**68.9**
Gammatone	power(0.1)	Before FB	69.6
Trapezoids in PLP	**power (0.33)**	**After FB**	**70.0**
Trapezoids in PLP	power (0.33)	Before FB	69.9

Moving the amplitude scaling to before the FB results in only a negligible decrement in performance. Table 2 varies the combinations of static/dynamic basis vectors, and numbers of dynamic terms. 13 static terms including log-energy was used with either filterbanks (Fig.3) or a Mel-shape warping (Fig.2) built into the unified basis vectors. The baselines are again bolded. A logarithmic nonlinearity before filterbanks was used.

Table 2. Phonetic accuracy (%) using different unified static/dynamic basis vectors

Static BV Type	Dynamic BV Type	Feature Num.	Accuracy
Mel+regular cosine	**delta, acceleration**	**39**	**69.9**
Gammatone+regular cosine	3 DCS	39	70.0
FFT+DCTC with mel warp	3 DCS	39	70.2
Mel+regular cosine	**delta, acc. & third order**	**52**	**70.1**
Gammatone+regular cosine	4 DCS	52	72.2
FFT+DCTC with mel warp	4 DCS	52	72.3
Gammatone+regular cosine	5 DCS	65	72.7
FFT+DCTC with mel warp	5 DCS	65	72.7

First, with the same amount of features, the combination of FFT+DCTC with Mel-shape warping and DCS cases are better than the bolded baselines (larger difference with 52 features). This is consistent with the finer frequency resolution reflected by the static basis vectors (compare Figure 2 and 3), and also better time resolution of the dynamic basis vectors (Figure 4). Also, note that if we compare the 39 and 52 feature cases, adding one more DCS basis vector brings much more significant improvement than adding one more differential basis vector. This again, shows that more DCS effectively provides finer temporal information.

3.2. Word level recognition task

In this section, we report word (actually character) level recognition to confirm the findings with the phonetic experiments. 37116 utterances spoken by 78 women speakers from the 863 Mandarin Chinese database were used as training data (about 40 hours in total), and another 5 women speakers (3125 utterances) were used as a test data. 16-mixture cross-word triphones and a 5868-word bigram model were trained for decoding. Throughout this section, we use character percentage accuracy as the evaluation measurement.

In Table 3, we repeated the cases in Table 1 to further confirm the validity of moving the amplitude scaling. The setup parameters for the frontends were identical to those in Table 1. The baselines are bolded.

Table 3. Character accuracy (%) of placing the amplitude scaling before/after the filterbanks

FB Type	Scaling Type	Scaling Position	Accuracy
Mel	**log**	**After FB**	**86.1**
Mel	log	Before FB	86.6
Mel	**power (0.1)**	**After FB**	**85.1**
Mel	power (0.1)	Before FB	85.8
Gammatone	**log**	**After FB**	**85.8**
Gammatone	log	Before FB	87.0
Gammatone	**power (0.1)**	**After FB**	**83.3**
Gammatone	power (0.1)	Before FB	85.9
Trapezoids in PLP	**power (0.33)**	**After FB**	**85.9**
Trapezoids in PLP	power (0.33)	Before FB	86.5

These results strengthen the validity of moving the amplitude scaling. In Table 4, we present two pairs of comparisons on different static/dynamic settings. The optimal frame length/space for the DCS scenarios were 10ms/2ms (for the Mel+regular cosine case) and 25ms/2ms (for the FFT+DCTC with Mel warping case). The optimal block length/space of DCS were 142ms/14ms for both. A logarithm scaling was placed after the filterbanks. Baselines are bolded.

Table 4. Character accuracy (%) using different unified static/dynamic basis vectors

Static BV Type	Dynamic BV Type	Feature Num.	Accuracy
Mel+regular cosine	**delta, acceleration**	**39**	**86.1**
Mel+regular cosine	6 DCS	78	86.7
FFT+DCTC with mel warp	**delta, acceleration**	**39**	**87.1**
FFT+DCTC with mel warp	6 DCS	78	87.8

Again, the FFT Mel warping is better than the filterbank Mel warping. The DCS is superior to differential dynamic basis vectors. We predict that with such high-dimensional features, the improvements would be more obvious with 32-mixture models, as shown in Table 2 for phonetic recognition.

4. CONCLUSIONS AND FUTURE WORK

In this work, we developed a unified framework by moving the amplitude scaling and modifying the basis vectors. Insights and useful applications were discussed in detail using examples. Extensive experiments confirmed the rearrangement of the nonlinearity. Also, various basis vector combinations were examined to show their determinant impacts on the system performance. Advanced frontend features will be developed based on this unified structure in our future work.

5. ACKNOWLEDGEMENT

This research is sponsored by the Air Force Research Laboratory under agreement number FA87501210093. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Lab or the U.S. Government.

6. REFERENCES

[1] J.S. Bridle and M.D. Brown, "An Experimental Automatic Word-Recognition System," *JSRU Report*, no.1003, Joint Speech Research Unit, Ruislip, England, 1974.

[2] J. Wu and J. Yu, "An Improved Arithmetic of MFCC in Speech recognition System," in *IEEE Int. Conf. Electro., Comm., and Control*, Sept.2011, pp.719-722.

[3] K. Kaewtip, L.N. Tan and A. Alwan, "A Pitch-Based Spectral Enhancement Technique for Robust Speech Processing," in *INTERSPEECH-2013*, Aug.2013, pp.3284-3288.

[4] M.A. Hossan, S. Memon and M.A. Gregory, "A Novel Approach for MFCC Feature Extraction," in *IEEE 4th Int. Conf. on Signal Processing and Communication Systems*, Dec.2010.

[5] S. Memon, M. Lech and N. Maddage, "Speaker Verification Based on Different Vector Quantization Techniques With Gaussian Mixture Models," in *Third Int. Conf. on Network and System Security*, 2009, pp.403-408.

[6] H.S. Jayanna and S.R.M. Prasanna, "Fuzzy Vector Quantization for Speaker Recognition under Limited Data Conditions," *TENCON 2008-IEEE Region 10 Conference*, 2008, pp.1-4.

[7] J. Chen, K.K. Paliwal, M. Mizumachi and S. Nakamura, "Robust MFCCs Derived From Different Power Spectrum," in *Eurospeech 2001*, Scandinavia, 2001.

[8] C. Wang, Z. Miao and X. Meng, "Differential MFCC and Vector Quantization Used for Real-Time Speaker Recognition System," in *IEEE Congress on Image and Signal Processing*, 2008, pp.319-323.

[9] S.A. Zahorian, H. Hu, Z. Chen and J. Wu, "Spectral and Temporal Modulation Features for Phonetic Recognition," in *INTERSPEECH-2009*, Sept. 2009, pp.1071-1074.

[10] F. Valente and H. Hermansky, "Hierarchical and Parallel Processing of Modulation Spectrum for ASR Applications," in *ICASSP-2008*, April 2008, pp.4165-4168.

[11] M. Kleinshmidt, "Localized Spectro-Temporal Features for Automatic Speech Recognition," in *Eurospeech 2003*, Sept. 2003, Switzerland.

[12] S. Seneff, "A Joint Synchrony/Mean Rate Model of Auditory Speech Processing," *Journal of Phonetics*, 16, pp.55-76, 1988.

[13] X. Zhang, M.G. Heinz, I.C. Bruce, and L.H. Carney, "A Phenomenological Model for the Response of Auditory-Nerve Fibers: I. Nonlinear Tuning With Compression and Suppression," *J.Acoust. Soc. Am.*, vol.109, no.2, pp.648-670, Feb.2001.

[14] C. Kim and R.M. Stern, "Feature Extraction for Robust Speech Recognition Using a Power-Law Nonlinearity and Power-Bias Subtraction," in *INTERSPEECH-2009*, Sept.2009, pp.28-31.

[15] H. Hermansky, "Perceptual Linear Prediction Analysis of Speech," *J. Acoust. Soc. Am.*, vol.87, no.4, pp.1738-1752, Apr, 1990.

[16] B.Meyer, S.V. Ravuri, M.R. Schadler and N. Morgan, "Comparing Different Flavors of Spectro-Temporal Features for ASR," in *INTERSPEECH-2011*, Aug.2011, pp.1269-1272.

[17] W. Ge, "Two Modified Methods of Feature Extraction for Automatic Speech Recognition," Master thesis, Department of Electrical and Computer Engineering, Binghamton University, Dec.2013.

[18] N. Mesgarani, G.S.V.S. Sivaram, S.K. Nemala, M. Elhilali and H. Hermansky, "Discriminant Spectrotemporal Features for Phoneme Recognition," in *INTERSPEECH-2009*, Sept.2009, pp.2983-2986.

[19] C.Kim and R.M. Stern, "Power-Normalized Cepstral Coefficients (PNCC) for Robust Speech Recognition," in *ICASSP-2012*, March 2012, pp.4101-4104.

[20] M. Slaney, "Auditory Toolbox Version 2," *Interval Research Corporation Technical Report*, no.10, 1998.

[21]. D. Ellis. (2006) PLP and RASTA (and MFCC, and inversion) in MATLAB using melfcc.m and invmelfcc.m. [Online].Available: http://labrosa.ee.columbia.edu/matlab/rastamat/

A2 X. Liu and S.A. Zahorian, Combined PNCC Feature Extractor for Robust Speech Recognition, *CHINASIP 2014*, August 2014.

COMBINED PNCC FEATURE EXTRACTOR FOR ROBUST SPEECH RECOGNITION

Xiaoyu Liu, Stephen A. Zahorian

Department of Electrical and Computer Engineering, Binghamton University,
Binghamton, NY 13902, USA

ABSTRACT

Recently, two major types of Power-Normalized Cepstral Coefficients (PNCCs) were proposed as noise robust Automatic Speech Recognition (ASR) front-end. All the literatures for these two PNCCs assume clean training data and clean or noisy test data. However, we find that one PNCC method has good performance for the clean training/noisy test scenario, but degrades when test data is cleaner than the training data. The other PNCC method performs relatively better for noisy training/clean test conditions, but is not very robust for the clean training/noisy test conditions. We propose Combined PNCC (C-PNCC) algorithm, which is superior to both previous PNCCs for clean training/noisy test cases, and which also has reasonably good performance for noisy training/clean test conditions.

Index Terms— C-PNCC, G-PNCC, L-PNCC, front-end, noise reduction

1. INTRODUCTION

Two major types of PNCC feature extractors were recently proposed in [1] and [2]. They have been proved to be superior to many other front-end features in many aspects. PNCCs obtain much better performance in dealing with non-stationary noise (such as background music) than Spectral Entropy based method in [3] and Voice Activity Detection based method in [4] since the spectral distribution of non-stationary noise cannot be as easily distinguished from speech spectrum as that of stationary noise. PNCCs also require much less computations while gain better accuracy than Vector Taylor Series algorithm in [5]. In addition, the power normalization part in PNCCs was also integrated into other front-end processing to improve the noise robustness, such as the Invariant-Integration Features (IIF) and Delta-Spectral Cepstral Coefficients (DSCC) front-ends in the work of [6] and [7], both of which use the noise estimation algorithms in PNCCs.

The detailed descriptions for the first PNCC is given in [1], and is abbreviated as G-PNCC in our work. This PNCC does not use noise reduction in the training phase. In testing, it iteratively subtracts time-constant noise estimate from a channel until the ratio of the Arithmetic Mean to Geometric Mean (AM-GM) of that channel achieves that of the training database. The second PNCC is described in [2], and is denoted as L-PNCC in our paper. L-PNCC uses the time-varying lower envelope of the power sequence as a noise estimate for each channel. This lower envelope is computed by an Asymmetric Noise Suppression (ANS) filter. The noise subtraction is applied to both training and testing data.

Both PNCCs were studied under clean training/noisy testing conditions in [1,2]. However, in practice, especially in ASR industry, it is not easy to always obtain clean training corpora. We found that G-PNCC generally had higher accuracy than L-PNCC under clean training/noisy testing conditions, whereas L-PNCC worked better along the other way. In this paper, we developed a Combined PNCC (C-PNCC) algorithm that combines and enhances the advantages of both G-PNCC and L-PNCC. Extensive experiments showed that C-PNCC was superior to both previous PNCCs for clean training/noisy testing conditions, and also achieved reasonably good performance for noisy training/clean test environment.

2. COMBINED PNCC

2.1. Training

Figure 1 is the block diagram of the training phase of C-PNCC. After a pre-emphasis filter and 256-point magnitude squared STFT, each 15ms frame is weighted by a 40-channel gammatone filter bank. The short time power output sequence of each channel *i* is mapped to the medium-duration power domain by averaging over $2M_1 + 1$ frames centered at the current frame according to equation (1):

$$Q(i,j) = \frac{1}{2M_1 + 1} \sum_{j'=j-M_1}^{j'=j+M_1} P(i,j') \quad (1)$$

where *i* is channel index, *j* the frame index, and *P(i,j)* the gammotone short time output power.

A weakness of G-PNCC is that it does not remove any noise in the training database. Thus, matching the AM-GM ratio of the test data, which is mathematically proved in [8] to be an increasing function of the channel SNR, to that of the training data degrades the performance if test data is cleaner than training data. To address this issue, we pre-process each training utterance by subtracting the lower envelope of each channel *i* in each training sentence. This lower envelope is computed by an Asymmetric Noise Suppression (ANS) filter, whose detailed description is given by [2]. Figure 2 depicts the 8th gammatone channel medium-duration power sequence and its lower envelope of a 0dB case (additive street noise). This lower envelope can be viewed as a simple version of the Minimum Statistics noise estimation algorithms in [9,10], where the estimation starts from the minimum power of a window and recursively adjusts its value by minimizing the estimation mean square error. As Figure 2 shows, the lower envelope tends to bias toward lower values. We will explain how to compensate this bias in the testing phase. As a preprocessing step, it has great simplicity. Furthermore, the amplitude of this estimate is time-varying. Experiments will show the advantage of this time-varying estimate relative to the constant noise estimate employed in G-PNCC. The residual sequence with negative

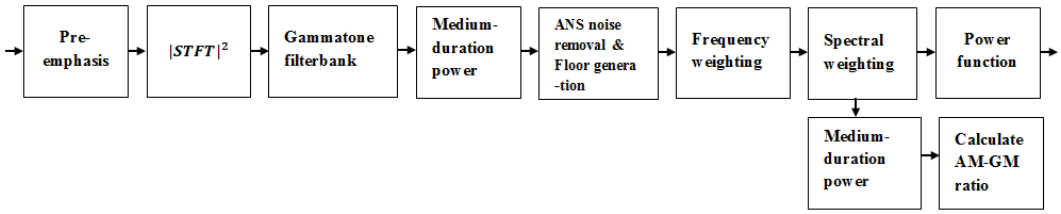

Fig. 1. Block diagram of C-PNCC training processing

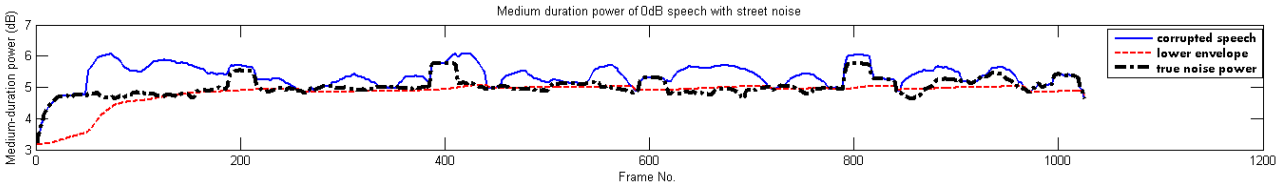

Fig. 2. Medium-duration power sequence, its lower envelope and true noise power of the 8th channel, 0dB speech corrupted by street noise. The true noise is obtained by eliminating the speech and computing the power of the noise only at the gammatone filter output

segments rectified to 0 is then processed by the ANS filter to create a noise floor. We denote the maximum between this floor and the residual sequence by $\tilde{Q}(i,j)$. Then, as in previous PNCCs [1,2], a frequency smoothed gain factor is computed by averaging over consecutive $2N_1+1$ channels for each fixed frame. This gain factor is used to convert the medium-duration power to short-time power according to equation (2), where *i* is the channel index, *j* the frame index.

$$\tilde{P}(i,j) = \left[\frac{1}{2N_1+1}\sum_{i'=i-N_1}^{i'=i+N_1}\frac{\tilde{Q}(i',j)}{Q(i',j)}\right]\cdot P(i,j) \qquad (2)$$

To cancel the bias of the initial noise estimation, we use the AM-GM ratio based noise searching algorithm in G-PNCC. The theoretical base of the AM-GM searching is that a noisy power sequence can be properly modeled by a gamma distribution [8], and a constant noise subtraction in G-PNCC does not change this power distribution. However, the time-varying noise removal in C-PNCC distorts this distribution. Consequently, the AM-GM ratio cannot properly reflect the SNR of each channel. To reduce this "distribution distortion", before computing the AM-GM ratio, a small portion of the original speech is added to the distorted distribution using a "spectral weighting factor" λ_s $(0<\lambda_s<1)$ according to the equation:

$$P_w(i,j) = \lambda_s P(i,j) + (1-\lambda_s)\tilde{P}(i,j) \qquad (3)$$

where $\tilde{P}(i,j)$ is the distorted power as in equation (2), and $P(i,j)$ is the original speech as in equation (1). The value of λ_s has significant impact on the recognition accuracy. Table 1 lists the recognition accuracy for TIMIT with different λ_s values under low SNR pink noise. The case $\lambda_s = 0$ corresponds to no spectral weighting. In this case, the distribution distortion causes degradation compared with direct G-PNCC, where there is no distribution distortion. In our work, a good value for λ_s is 0.05.

Each $P_w(i,j)$ is then power function amplitude scaled with a power value 1/15, and is output to the DCTC step. In parallel, the $P_w(i,j)$'s are converted back to the medium-duration power domain according to equation (1), except that we use $2M_2+1$ frames this time. The AM-GM ratio *G(i)* is then computed for each channel *i* according to equation (4), where $\Omega(i,j)$ is the medium duration power; *J* is the total number of frames. Finally, a $G_{train}(i)$ is obtained by averaging over *G(i)*'s of all utterances and stored as the average SNR level of the training set for channel *i*. In our work, we use $M_1 = 4,\ M_2 = 10$.

$$G(i) = \ln\left[\frac{1}{J}\sum_{j=1}^{J}\Omega(i,j)\right] - \frac{1}{J}\sum_{j=1}^{J}\ln\Omega(i,j) \qquad (4)$$

Table 1 Recognition accuracy with different λ_s values under low SNR pink noise

	10dB	5dB	0dB
$\lambda_s = 0$	48.5	37.1	25.9
$\lambda_s = 0.05$	52.5	43.7	33.3
$\lambda_s = 0.55$	50.7	41.0	30.1
G-PNCC	49.9	40.4	29.2

2.2. Testing

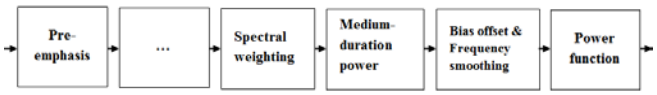

Fig. 3. Block diagram of C-PNCC test phase

Figure 3 depicts the block diagram of the test phase. It is very similar to the training steps in Figure 1. So, we use "…" to skip some of the identical blocks. The key step in the testing phase is to offset the bias of the initial noise subtraction for each test utterance. For each channel *i*, we start from -10dB relative to the minimum power of that channel, and if the *G*(*i*) value becomes larger than $G_{train}(i)$ after this noise estimate is assumed to have been subtracted, this estimate is regarded as the final compensation for the bias; otherwise, we increase the estimated noise level by 1dB and repeat this processing. The final noise adjustment is subtracted after the iterations are done, and the residual is frequency smoothed using equation (5), where $P_G(i,j)$ is the short time power which is then power function scaled; $\widetilde{\Omega}(i,j)$ is the medium duration power after making the final noise adjustment. In our work, the values for N_1 and N_2 are $N_1 = 3$, $N_2 = 8$. Figure 4 shows the overall estimate of the noise with the bias cancelled for a test sentence.

$$P_G(i,j) = \left[\frac{1}{2N_2+1}\sum_{i'=i-N_2}^{i'=i+N_2}\frac{\widetilde{\Omega}(i',j)}{\Omega(i',j)}\right]\cdot P_w(i,j) \qquad (5)$$

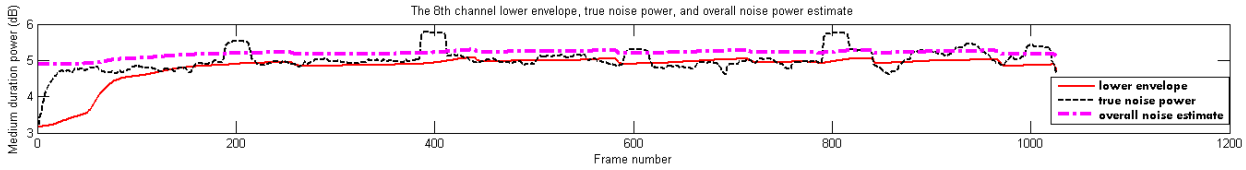

Fig. 4. The lower envelope, true noise power, and the overall noise estimate of the 8th channel. The speech is corrupted by 0dB street noise. The overall estimate consists of a time-varying initial estimate and a constant offset of the bias made by AM-GM searching.

3. EXPERIMENTAL RESULTS

To evaluate different PNCCs, experiments were conducted using TIMIT for a 39 phoneme recognition task. 3696 and 1344 sentences were used for training and testing respectively. 13 Discrete Cosine Transform Coefficients (DCTCs) and 6 Discrete Cosine Series Coefficients (DCSCs) as suggested in [11,12] were used to encode the spectrum after PNCC processing. The baseline MFCC used 39 features (including delta and acceleration terms). A 3-state HMM model with 32 Gaussian mixtures was built for each phone. The training and decoding were both run with HTK 3.4. For better comparisons, all the parameters in the three PNCC methods were optimized.

3.1. Stationary noise

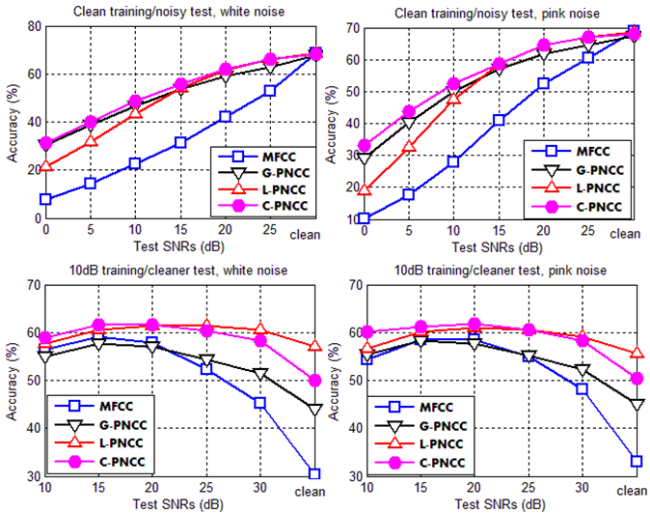

Fig. 5. Comparison of PNCCs for stationary noise. Top: clean training/noisy test cases. Bottom: 10dB training/cleaner testing cases. Left: white noise. Right: pink noise.

First, two stationary noises were added respectively. The results are presented in Figure 5. The left panel is for white noise, and the right panel is for pink noise. The top panel is for training on clean speech, and testing varying from clean speech to 0dB; the bottom panel is for training on 10dB noisy speech, and testing from 10dB to clean speech. The vertical axis is the recognition accuracy and the horizontal axis represents the SNRs of the test data. From the top panel, C-PNCC has the highest accuracy for clean training/noisy test conditions, with G-PNCC the second best. This shows the time-varying noise estimate is superior to a constant noise estimate, even for stationary noise. Notice that though G-PNCC is significantly better than L-PNCC under clean training/noisy testing conditions, especially for low SNRs, it degrades dramatically along the other way in the bottom panel. However, C-PNCC shows improvements for clean training conditions (top panel) while remains almost the same performance as L-PNCC for noisy training/clean testing environment. In the bottom panel, the average degradation of C-PNCC relative to L-PNCC is less than 1% for each type of noise.

3.2. Non-stationary noise

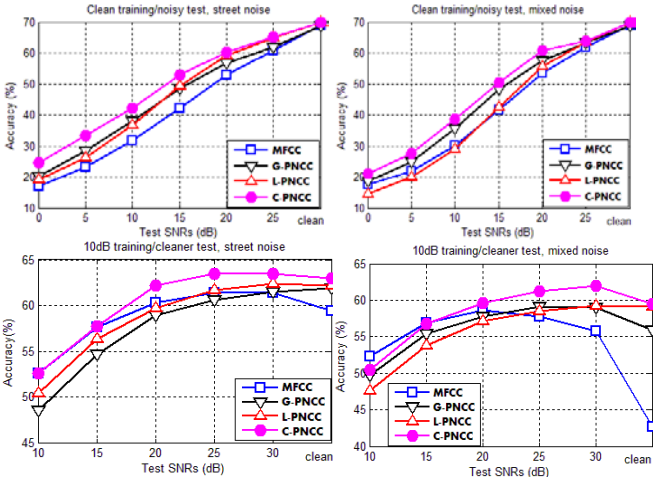

Fig. 6. Comparison of PNCCs for non-stationary noise. Top: clean training/noisy test cases. Bottom: 10dB training/cleaner testing cases. Left: street noise. Right: mixed noise.

Figure 6 presents the results for two types of non-stationary noise that are widely encountered for most ASR applications. The left panel is for street noise, which was recorded during peak time in a street in New York City. The right panel is for "mixed noise", which contains about 60 interfering speakers as well as background music in a bar. The top panel is again clean training/noisy testing settings, and the bottom panel for 10dB training/cleaner testing scenario. Similarly to the previous cases, C-PNCC has the best performance under clean training conditions. Meanwhile, it also provides superior performance when test data is cleaner than training data (bottom panel). It is also noticed that though L-PNCC performs better than G-PNCC in the bottom panel, they both degrade compared with the baseline MFCC front-end for some test SNRs. However, C-PNCC provides solid improvement relative to the baseline MFCC front-end.

3.3. Comparison with other front-ends

In this section, we present a brief comparison among C-PNCC, gammatone+power amplitude scaling, RASTA-PLP and baseline MFCC front-end. The comparison was conducted for clean training/noisy test setting using the two non-stationary noises in Section 3.2. Both the baseline MFCC and RASTA-PLP front-ends

use 39 features. The RASTA-PLP front-end is an implementation of the algorithm described in [13].

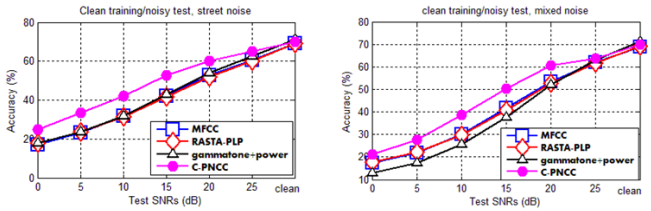

Fig. 7. Comparison of C-PNCC with baseline MFCC, RASTA-PLP and gammatone+power front-end for clean training/noisy testing conditions. Left: street noise, Right: mixed noise.

Figure 7 depicts the results. The left panel is for street noise, and the right panel is for mixed noise. The MFCC and RASTA-PLP front-end have very close performance, and C-PNCC has significant improvements compared with the other three front-ends. It is interesting to see that the gammatone+power method did not provide improvement compared with the baseline MFCC. It is even worse in the mixed noise case.

In the next experiment, we kept the mel filter bank structure in the baseline MFCC, but replaced the 39 features with 78 spectral/temporal features (13 DCTC/6DCSC), which is the same set of features used in the gammatone+power front-end. We also replaced the logarithmic amplitude scaling in the MFCC with a power-law nonlinearity. The power value used was 1/15 in both MFCC and gammatone front-ends. Again, we plot the accuracy of the mel+power and gammatone+power front-ends in Figure 8.

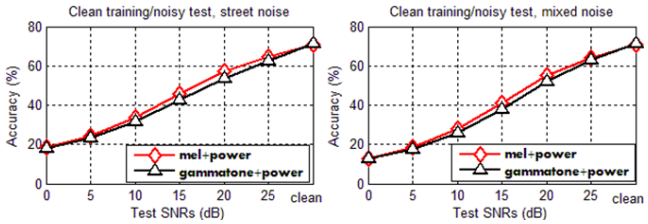

Fig. 8. Comparison of mel+power and gammatone+power front-ends in the presence of street noise (left) and mixed noise (right) under clean training/noisy testing conditions.

The result shows that the mel+power front-end constantly performs better than the gammatone+power front-end for the two non-stationary noises at all SNR levels. Since the only difference in these two front-ends is the filter bank type, a tentative reason is that the mel triangular filter shape has sharper onset than the gammatone filter, thus may render better characterization of the temporal masking effect. This is based on the theory in [14,15] that human ears tend to focus more on the onset of the power envelope than on the falling edge. We also found (not shown in this work) that in the presence of stationary noise, such as white noise or pink noise, the same experiment had different results: the gammatone+power front-end performed constantly better than the mel+power front-end. This might be because the temporal masking effect in the presence of stationary noise is not as strong as it is with non-stationary noise.

Based on this idea, the final experiment is to compare the performance of mel C-PNCC with gammatone C-PNCC. We implemented the C-PNCC noise reduction algorithm to 26 mel channels in place of the 40 gammatone channels. Since there were fewer channels needed, the frequency smoothing parameter N_2 in equation (5) was reduced to 3; the other parameters remained unchanged ($M_1 = 4, N_1 = 3, M_2 = 10, \lambda_s = 0.05$). The experiment configuration was the same as Section 3.2.

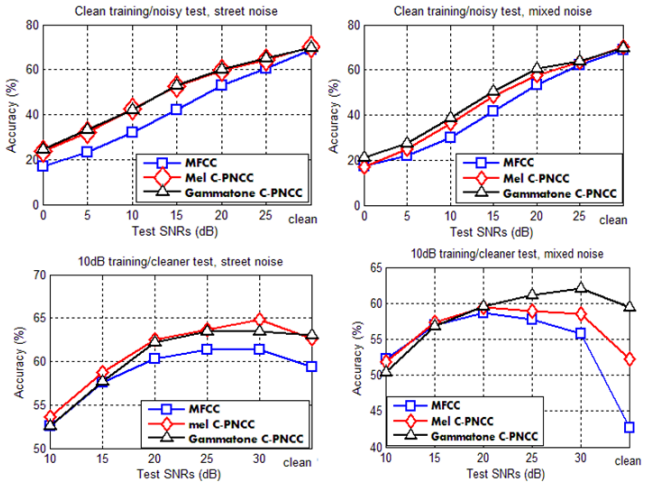

Fig. 9. Comparison of mel C-PNCC with gammatone C-PNCC. Left panel: street noise. Right panel: mixed noise. Top panel: clean training/noisy testing. Bottom panel: 10dB training/cleaner testing.

The results are listed in Figure 9. The mel C-PNCC has very similar performance as the gammatone C-PNCC. Compared with the baseline MFCC, which does not have noise reduction part, the mel C-PNCC obtains substantial improvements both for clean training/noisy test and 10dB training/cleaner test scenarios.

According to this result, using mel filterbank can reduce computational load in PNCC processing, while maintain the performance. Since all PNCCs operate on a channel-by-channel basis, using 26 mel frequency channels can reduce the run time by approximately 1/3 compared with 40 gammatone channels.

4. CONCLUSIONS

In this paper, we proposed a C-PNCC algorithm which combines and enhances the advantages of G-PNCC and L-PNCC. Extensive experiments were conducted, including stationary noise and non-stationary noise. Both clean training/noisy testing and noisy training/cleaner testing conditions were investigated. C-PNCC showed promising performance in all cases. In addition, a comparison of PNCC front-end with other front-ends was conducted. The same advantage of C-PNCC was also verified by mel filter bank, which had comparable performance as gammatone filter bank, but required less computations.

5. ACKNOWLEDGEMENT

This research is sponsored by the Air Force Research Laboratory under agreement number FA87501210093. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Laboratory or the U.S. Government.

6. REFERENCES

[1] C. Kim and R.M. Stern, "Feature Extraction for Robust Speech Recognition using a Power-Law Nonlinearity and Power-Bias Subtraction," in *INTERSPEECH-2009*, pp. 28-31, Sept. 2009.

[2] C. Kim and R.M. Stern, "Power-Normalized Cepstral Coefficients (PNCC) for Robust Speech Recognition," in *ICASSP 2012*, March. 2012.

[3] H. Mirsa, S. Ikbal, H. Bourlard, and H. Hermansky, "Spectral Entropy Based Feature for Robust ASR," in *IEEE Int. Conf. Acoust. Speech, and Signal Processing*, pp 193-196, May. 2004.

[4] D.-S. Kim, S.-Y. Lee, and R.M. Kil, "Auditory Processing of Speech Signals for Robust Speech Recognition in Real-world Noisy Environments," *IEEE Trans. Speech and Audio Processing*, vol. 7, no.1, pp. 55-69, 1999.

[5] P.J. Moreno, B. Raj, and R.M. Stern, "A Vector Taylor Series Approach for Environment-independent Speech Recognition," in *IEEE Int. Conf. Acoust., Speech and Signal Processing*, pp. 733-736, May. 1996.

[6] F. Müller, A. Mertins, "Noise Robust Speaker-independent Speech Recognition With Invariant-integration Features Using Power-bias Subtraction," in *INTERSPEECH-2011*, pp. 1677-1680, Aug. 2011.

[7] K. Kumar, C. Kim, R.M. Stern, "Delta-spectral Cepstral Coefficients for Robust Speech Recognition," in *IEEE Int. Conf. on Acoust. Speech, and Signal Processing*, pp. 4784-4787, May 2011.

[8] C. Kim and R.M. Stern, "Robust Signal-to-Noise Ratio Estimation Based on Waveform Amplitude Distribution Analysis," in *INTERSPEECH*-2008, pp. 2598-2601, Sept. 2008.

[9] R. Martin, "Noise Power Spectral Density Estimation Based on Optimal Smoothing and Minimum Statistics," in *IEEE Trans. on Speech and Audio Processing*, vol. 9, no.5, pp. 504-512, July 2001.

[10] S. Seyedtabaee, H.M. Goodarzi, "Improved Noise Minimum Statistics Estimation Algorithm for Using in a Speech-passing Noise-rejecting Headset," in *EURASIP Journal on Advances in Signal Processing*, 2010.

[11] S.A. Zahorian, H. Hu, Z. Chen, and J. Wu, "Spectral and Temporal Modulation Features for Phonetic Recognition," in *INTERPEECH-2009*, pp. 1071-1074, Sept. 2009.

[12] V.N. Parinam, C. Vootkuri, and S.A. Zahorian, "Comparison of Spectral Analysis Methods for Automatic Speech Recognition," in *INTERSPEECH-2013*, pp. 3356-3359, Sept. 2013.

[13] H. Hermansky and N. Morgan, "RASTA Processing of Speech," in *IEEE. Trans. Speech Audio Process.*, vol.2, no. 4, pp. 578-589, Oct. 1994.

[14] C. Lemyre, M. Jelinek, R. Lefebvre, "New Approach to Voiced Onset Detection in Speech Signal and Its Application for Frame Error Concealment," in *IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, pp. 4757-4760, May 2008.

[15] T.S. Gunawan, E. Ambikairajah, "A New Forward Masking Model and Its Application to Speech Enhancement," in *IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, pp. 149-152, May 2006.

A3 H. Hu, S. A. Zahorian, P. Guzewich, and J. Wu, "Acoustic Features for Robust Classification of Mandarin Tones," *Interspeech 2014*, September 2014**.**

ACOUSTIC FEATURES FOR ROBUST CLASSIFICATION OF MANDARIN TONES

Hongbing Hu[1], *Stephen A. Zahorian*[1], *Peter Guzewich*[1], *Jiang Wu*[1]

[1] Department of Electrical and Computer Engineering, Binghamton University, Binghamton, NY, 13902, USA
{hongbing.hu, zahorian, peter.guzewich, jiang.wu}@binghamton.edu

ABSTRACT

For applications such as tone modeling and automatic tone recognition, smoothed F_0 (pitch) all-voiced pitch tracks are desirable. Three pitch trackers that have been shown to give good accuracy for pitch tracking are YAAPT, YIN, and PRAAT. On tests with English and Japanese databases, for which ground truth pitch tracks are available by other means, we show that YAAPT has lower errors than YIN and PRAAT. We also experimentally compare the effectiveness of the three trackers for automatic classification of Mandarin tones. In addition to F_0 tracks, a compact set of low-frequency spectral shape trajectories are used as additional features for automatic tone classification. A combination of pitch trajectories computed with YAAPT and spectral shape trajectories extracted from 800ms intervals for each tone results in tone classification accuracy of nearly 77%, a rate higher than human listeners achieve for isolated tonal syllables, and also higher than that obtained with the other two trackers.

Index Terms: pitch tracking, tone classification, Mandarin Chinese, fundamental frequency

INTRODUCTION AND BACKGROUND

Accurate fundamental frequency (F_0) (commonly referred to as pitch—the terms pitch and F_0 are used interchangeably in this paper) tracking in speech remains an elusive goal, especially for noisy and/or band-limited speech, typically the scenarios where reliable pitch tracking would be most useful. Good results have been reported by Talkin in RAPT where a normalized cross correlation function is used 0. High accuracy pitch tracking results have also been obtained by the YIN algorithm, which uses a modified version of the autocorrelation method 0. Probably the most widely used tool for pitch tracking is the speech analysis program PRAAT [1] because it provides fairly reliable tracking and is readily available. Since about 1980, several pitch trackers have been developed and several studies have been done to evaluate these trackers [5, 7]. Our own tool for pitch tracking is named YAAPT for "Yet Another Algorithm for Pitch Tracking".

For automatic recognition of tones in tonal languages such as Mandarin, robust all-voiced pitch tracking is especially important, as pitch is widely considered as the most important acoustic correlate of a tone.

In this paper, we first summarize and illustrate the YAAPT method in the remainder of this section. Section 2 introduces several modifications motivated by the desire to improve automatic tone classification and describes a method for computing spectral temporal features, which are effective in addition to pitch for use in tone classification. The evaluation results of several experiments, which illustrate the effectiveness of YAAPT and the additional features, are reported in Sections 3 and 4. For control purposes, experimental results obtained with YIN and PRAAT pitch trackers are also given.

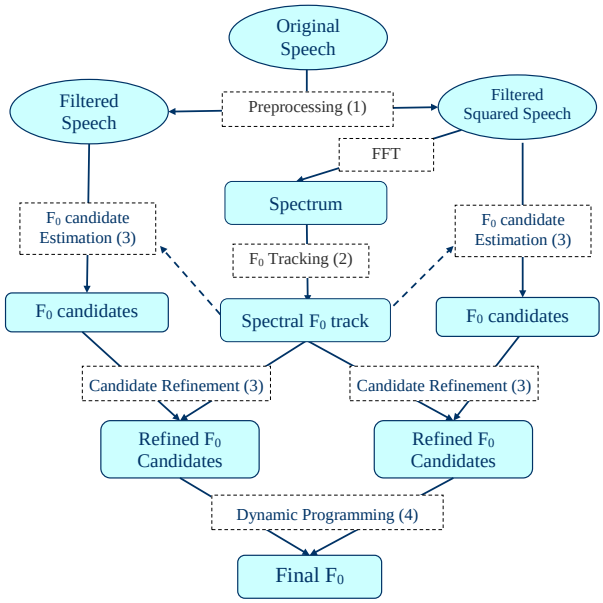

Figure 1: *YAAPT flow chart*

The main signal processing steps in YAAPT are illustrated in figure 1. For each frame of speech, multiple pitch candidates are computed using the normalized cross correlation. A smoothed pitch track is computed from the spectrogram of the squared signal; to some extent the squaring restores the fundamental, which is likely to be missing from band-limited speech such as telephone speech. All F_0 candidates, both time domain and frequency domain, as well as an unvoiced candidate, are assigned merit values and the highest overall merit path is determined using dynamic programming. More details, as well as illustrations of the various steps involved, are given in [12]. All three of these trackers have settings to minimize "Gross" error (large errors in the voiced sections of speech) or "Big" error, which takes into account both large errors in the voiced speech regions, and voiced/unvoiced decision errors. Unfortunately, neither of these minimum error cases is best suited for computing pitch tracks for Mandarin tone classification.

ALGORITHMS

YAAPT improvements

The most significant change is the introduction of additional post-processing techniques to refine the final pitch tracks, especially for the case when the track is intended to be all voiced. With the previous settings, as given in [12], optimized for minimum gross error, visual inspection of computed pitch tracks showed apparent abnormalities, especially in the interpolated pitch values through unvoiced regions. Nevertheless, the gross error values for YAAPT were low, since the estimated pitch values in actual unvoiced regions were not considered in the error calculation.

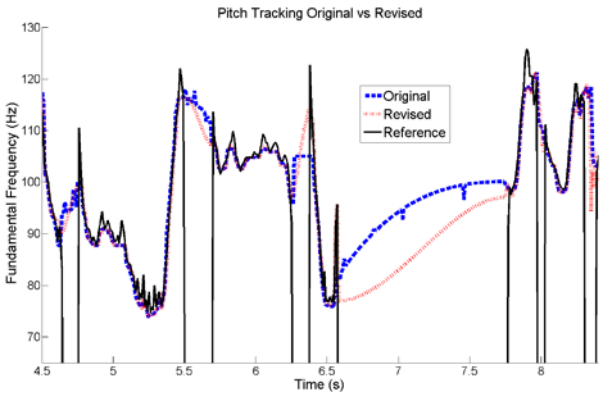

Figure 2: *Illustration of YAAPT F0 tracking. Original (Blue), revised (Red), ground truth reference (Black).*

To improve YAAPT, the algorithm was changed and now always determines the minimum big error track with voicing decisions, even if finally an all voiced (minimum gross error) track is desired. Heuristics are then incorporated to identify and eliminate pitch values which appear to be in error due to pitch halves or doubles. If a track with minimum gross error is desired, post processing then includes cubic polynomial interpolation through the unvoiced regions using a filtered version of the calculated track. This method was empirically determined to work effectively at reducing error and producing a smooth track. Figure 2 above depicts the ground truth pitch track, the former YAAPT track, and the YAAPT track with the modifications introduced in this paper.

The last of the modifications to YAAPT was a code refinement to improve the processing time and accuracy. One of the more significant of these modifications was to change an inner loop for the spectral harmonic correlation calculation to reduce computational time. Two other changes to this section of code helped improve overall performance by more accurately calculating the spectral track, even with a shorter FFT length for spectral calculations. These changes corrected for possible frequency misalignment between temporal and spectral pitch candidates, which depended on the frequency resolution (FFT length). Consequently, a shorter FFT length can be used, decreasing computational time, while not significantly degrading performance. These code refinements decreased overall computation time by around 25% and decreased error rates by small percentages.

Additional spectral temporal features useful for tone classification

In our initial work with Mandarin tone recognition 0, we observed that the four primary Mandarin tones (High, Rising, Falling, Dipping) were also relatively apparent from inspection of the low frequency region of the spectrogram. Therefore, global spectral shape trajectories, computed with a small number of spectral Discrete Cosine Transform Coefficients (DCTCs) each of which is encoded with several Discrete Cosine Series Coefficients (DCSCs), appeared to be a relatively effective approach for computing tone features. The details of DCTC and DCSC calculations are given elsewhere [13]. Summarizing briefly, DCTCs are coefficients of a cosine-like basis vector expansion of speech log magnitude spectra, where the cosine basis vectors are modified to take into account a mel-like frequency scale. A DCTC representation of speech spectra is a smoothed representation, with degree of smoothing determined by the number of DCTCs used. A DCSC encoding of any feature over time (such as a DCTC term or pitch) is a cosine basis vector expansion over time, but with the cosine basis vectors modified to give more resolution near the center of the time interval and less resolution near the endpoints of the interval. In our work, the time resolution of a DCSC representation was determined by a Kaiser window, with the degree of resolution variation determined by the Kaiser constant. The DCTCs/DCSCs are similar to MFCCs and delta/acceleration terms 0, but more general and flexible.

EXPERIMENTAL EVALUATIONS OF PITCH TRACKING ACCURACY

In order to evaluate the accuracy of YAAPT for pitch tracking accuracy, pitch tracks were computed from two databases, the Keele pitch database 0 and a Japanese database 0, for which ground truth pitch tracks are available. The Keele database contains 10 sentences, each about 30 seconds long, with each sentence spoken by a different British speaker. Both studio quality and telephone versions of the speech were used. The Japanese database consists of 30 utterances by 14 male and 14 female speakers, resulting in a total of 840 utterances.

The pitch tracks were computed and compared using YAAPT, YIN, and PRAAT for both full bandwidth and telephone and/or simulated telephone conditions. Tests were done with clean versions of the speech and also at 5 dB SNR levels with additive white noise and additive babble noise. For YAAPT and PRAAT, tracks were computed both for an all-voiced condition and a condition for which the tracker made voiced/unvoiced decisions. For YIN, the track is always considered to be all-voiced, so that was the only case tested.

Results in terms of Big Error and Gross Errors are given in tables 1, 2, and 3 for clean speech, white noise at a 5 dB SNR (W-5), and babble noise at a 5 dB SNR (B-5).

Table 1:. Big and Gross errors (%) with the Keele database

		Studio			Simulated telephone		
	Tracker	Clean	W-5	B-5	Clean	W-5	B-5
Gross Error	YAAPT	3.1	3.4	7.9	4.6	6.4	28.2
	PRAAT	5.2	7.8	17.3	11.2	14.3	29.8
	YIN	3.0	4.6	14.8	21.0	27.3	38.5
Big Error	YAAPT	6.1	8.1	21.7	14.0	16.8	43.8
	PRAAT	8.7	19.9	34.2	15.4	21.3	47.5

Table 2:. Errors (%) with Keele telephone speech

		Telephone		
	Tracker	Clean	W-5	B-5
Gross Error	YAAPT	4.9	9.4	23.8
	PRAAT	12.6	22.9	31.2
	YIN	14.0	26.3	35.0
Big Error	YAAPT	9.9	20.5	45.9
	PRAAT	16.3	30.1	44.6

Table 3:. Big and Gross errors (%) with the Japanese database

		Studio			Simulated telephone		
	Tracker	Clean	W-5	B-5	Clean	W-5	B-5
Gross Error	YAAPT	1.8	2.9	4.0	4.4	7.3	24.4
	PRAAT	4.1	5.9	15.4	6.4	11.2	28.7
	YIN	1.7	2.9	13.0	14.5	21.0	34.5
Big Error	YAAPT	5.0	7.2	15.3	12.2	17.1	35.1
	PRAAT	7.1	17.9	31.0	10.1	23.3	43.8

For the clean full bandwidth conditions, the errors are small and fairly similar for all three pitch tracker methods. However, for most of the noisy and or band-limited cases, YAAPT results in lower error rates than for the other two trackers. For example, for the case of Keele telephone speech, and additive white noise, the gross error for YAAPT is under 10%, whereas for the other two pitch trackers, it is over 20%. Note that for comparable cases tested, error values are quite similar to those obtained in [12]; although YAAPT was "improved," the changes are more apparent by visual inspection of the tracks. Big and Gross error figures changed very little.

EXPERIMENTAL EVALUATIONS OF TONE CLASSIFICATION

Although YAAPT gives lower error rates than either YIN or PRAAT, it was still not clear which tracker would be the most effective for Mandarin tone classification. Therefore a series of tone classification experiments, comparing the three trackers, was performed.

The database used was the Shanghai region portion of RASC863 0. Only the four prominent tones of Mandarin (H, R, F, D) were used. Tone labels supplied with RASC863 were considered as ground truth. A multilayer feed-forward neural network classifier, trained with back propagation, was used for classifying tones from a combination of pitch and/or DCTC/DCSC spectral features. The number of network inputs ranged from 7 to 42, as described below, depending on the feature set under evaluation. In all cases, the network had 50 hidden nodes in the first hidden layer, 25 nodes in the second hidden layer, and 4 output nodes (one for each of the four tones). The overall configuration of the network (with two hidden layers with sigmoidal nonlinearities and number of nodes mentioned) was determined from pilot tests. A total of 1539 sentences were used for training; 670 sentences were used for testing.

Five feature conditions were tested in conjunction with each pitch tracking method: Spectral trajectory features only (35 features): computed with 5 DCTC terms each encoded with 7 DCSC terms, from a frequency range of 50 to 800 Hz. These particular conditions are consistent with observations of spectrograms that indicate tonal information is most easily observed in the low frequency region over segments longer than 100 ms.

1. "Raw" pitch trajectories (P) (7 features): encoded with 7 DCSC terms.
2. Normalized pitch trajectories (NP) (7 features): also each encoded by 7 DCSC terms. The normalization is accomplished by first computing the mean and standard deviation of the pitch over the entire sentence from which each tone segment is extracted. These mean values are then subtracted from pitch values in each segment, and the resultant values divided by the standard deviation.
3. A combination of feature sets 1 and 2. (42 features)
4. A combination of feature sets 1 and 3. (42 features)

In addition to testing each of the five feature cases above, for each of the three pitch trackers (15 conditions), since tones clearly have a temporal aspect, four different segment lengths were evaluated for classifying tones: 100 ms, 200 ms, 400 ms, and 800 ms. For each of these cases, segments were selected with a midpoint equal to the midpoint of the labeled tone. For the longer segment lengths, undoubtedly the segments extended into following and/or proceeding tones. However, this additional context was found to be somewhat beneficial, as shown in the following results. Results, in terms of tone classification accuracy, are given in figure 4 for YAAPT, YIN, and PRAAT with feature set 1 (DCTC), set 2(P), set 3(NP), set 4 (DCTC+P) and set 5 (DCTC+NP).

Several conclusions can be drawn about the tone classification results:

1. The features obtained with YAAPT result in considerably higher tone classification accuracy than for pitch features obtained with the other two trackers except for the shortest segment length tested (100 ms). The highest accuracy obtained with YAAPT based pitch tracks (76.9%) is 4.1% higher than the highest accuracy obtained with YIN and 5% higher than the best result obtained with PRAAT, and higher than the accuracy of humans for recognizing context-free tones [8] (~75%).

2. Although pitch features are most important for tone classification, the addition of spectral shape trajectory features improves accuracy by about 5%.
3. Pitch normalized features are more effective than raw pitch features, at least for YAAPT for shorter segment lengths. For the case of YIN and PRAAT, and YAAPT for long segment lengths, pitch normalization doesn't appear to be beneficial.
4. Tone classification accuracy improves as segment length increases, showing the importance of the long temporal variation.

We hypothesize that YAAPT is superior to both YIN and PRAAT for Mandarin tone classification primarily because of the better interpolation through unvoiced regions as illustrated in figure 3, where PRAAT and YIN can be seen to exhibit large anomalies compared to YAAPT, primarily in the unvoiced regions. Although details are not given here, due to length constraints, the previous version of YAAPT (as in [12]) resulted in tone classification accuracies typically 1% to 7% lower than for the YAAPT results reported here.

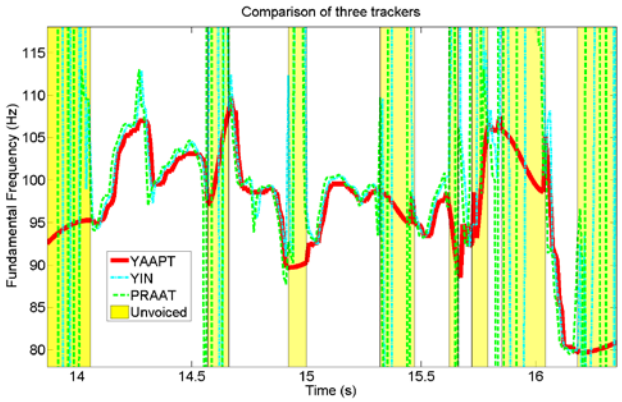

Figure 3: *Comparison of trackers. Highlighted (unvoiced) portions show large anomalies for PRAAT and YIN.*

CONCLUSIONS

This paper presents several modifications to YAAPT including a smooth interpolation of pitch through unvoiced regions with the interest of improving pitch modeling for Mandarin tones. The experiments demonstrate that YAAPT has lower errors, especially for noisy bandlimited speech, than either YIN or PRAAT pitch trackers. The YAAPT features, when combined with DCTC/DCSC features to capture spectral-temporal trajectories, are also shown to be more effective than either YIN or PRAAT pitch features.

The YAAPT algorithm is available at http://www.ws.binghamton.edu/zahorian/yaapt.htm as a MATLAB function, along with a user guide and recommendations for parameter settings. We have begun a series of character recognition experiments with continuous Mandarin to more thoroughly compare the effects of different pitch

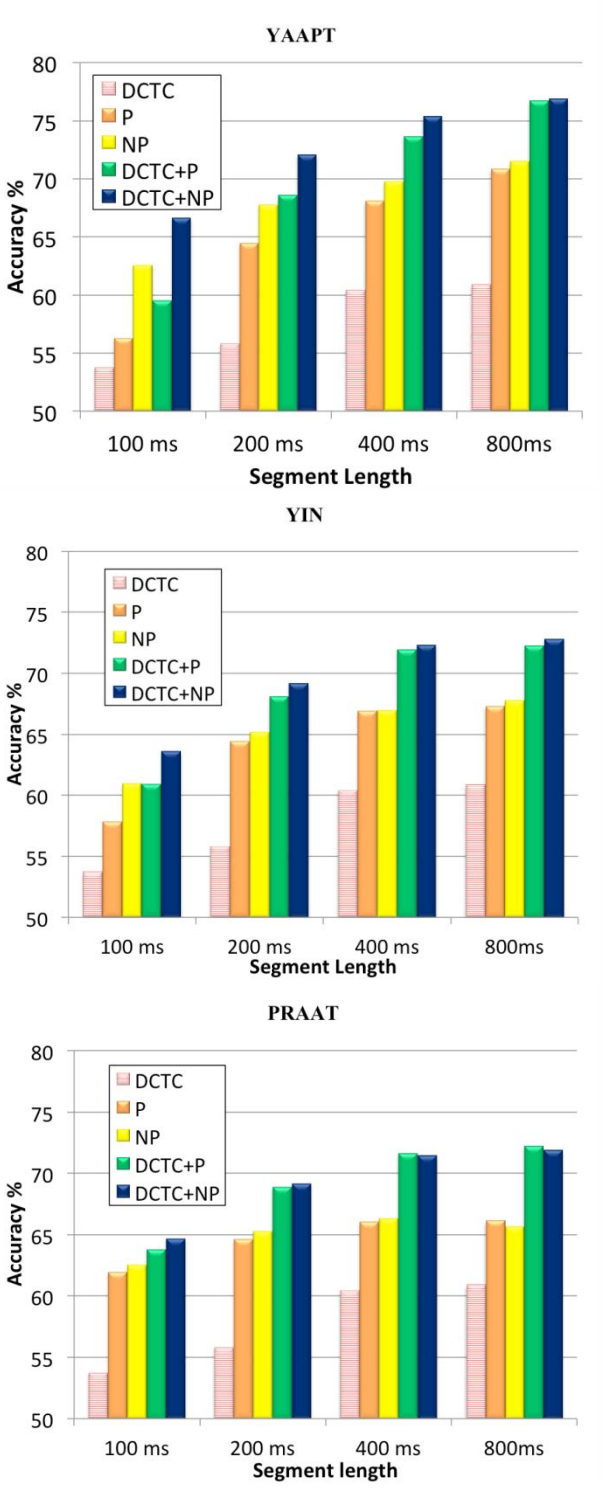

Figure 3: *Tone classification accuracy for features based on YAAPT (top), YIN (middle) and PRAAT (bottom).*

ACKNOWLEDGEMENT

This research is sponsored by the Air Force Research Laboratory under agreement number FA87501210093. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Laboratory or the U.S. Government.

REFERENCES

Boersma, P., and Weenink, D. (2001). "PRAAT, a system for doing phonetics by computer," *Glot International 5*(9/10), 341-345.

de Cheveigne, A., and Kawahara, H. (2002). "YIN, a fundamental frequency estimator for speech and music," *J. Acoust. Soc. Am.* 111(4), 1917–1930.

Liu, Z., Zhang, P., Shao, J., Zhao, Q., Yan, Y. and Feng, J. (2007), "Tone recognition in Mandarin spontaneous speech." *Proc. of the 4th International Conference on Non-Linear Speech Processing* (NOLISP 2007.)

Li, A., and et al. (2004), "RASC863-A Chinese speech corpus with four regional accents," *Chinese Academy of Social Sciences technical report.*

Mousset, E., Ainsworth, W. A., and Fonollosa, J. A. R. (1996). "A comparison of several recent methods of fundamental frequency and voicing decision estimation," in *Fourth Int. Conf. on Spoken Language Processing* (ICSLP, Philadelphia, Pennsylvania), pp.1273–1276.

Plante, F., Meyer, G., and Ainsworth, W. A. (1995). "A pitch extraction reference database," in *Fourth European Conf. on Speech Communication and Technology (EUROSPEECH, Madrid, Spain)*, pp. 837–840.

Rabiner, L., Cheng, M., Rosenberg, A., and McGonegal, C. (1976). "A comparative performance study of several pitch detection algorithms," *IEEE Trans. On Acoustics, Speech, and Signal Processing*, ASSP-24(5), 399–418.

Talkin, D. (1995). "A robust algorithm for pitch tracking (RAPT)," in *Speech Coding and Synthesis*, edited by W. B. Kleijn and K. K. Paliwal (Elsevier Science Publishers B.V., New York), pp. 495–518.

Wang, C., and Seneff, S. (2000). "Robust pitch tracking for prosodic modeling in telephone speech," in *IEEE Int. Conf. on Acoustics, Speech, and Signal Processing* (*ICASSP*, Istanbul, Turkey).

Wu, J., Zahorian, S. A., and Hu, H. (2013). "Tone recognition for continuous accented Mandarin Chinese," *ICASSP* 2013.

Zahorian, S. A., Hu, H., Chen, Z., and Wu, J. (2009) "Spectral and temporal modulation features for phonetic recognition," *INTERSPEECH* 2009.

Zahorian, S. A., and Hu, H. (2008) "A spectral/temporal method for robust fundamental frequency tracking," J. Acoust. Soc. Am. 123 (6), pp. 4559-4571. *Acoustics, Speech, and Signal Processing*, ASSP-24(5), 399–418.

Zahorian, S. A., Hu, H., Chen, Z., Wu, J., "Spectral and Temporal Modulation Features for Phonetic Recognition," in INTERSPEECH 2009, Sept. 2009.

A4 V. N. Parinam, C. Vootkuri, and S. A. Zahorian, "Comparison of Spectral Analysis Methods for Automatic Speech Recognition," *Interspeech 2013*, September 2013

COMPARISON OF SPECTRAL ANALYSIS METHODS FOR AUTOMATIC SPEECH RECOGNITION

Venkata Neelima Parinam, Chandra Vootkuri, Stephen A. Zahorian

Department of Electrical and Computer Engineering, Binghamton University, Binghamton, NY 13902, USA
{vparina1, cvootku1, zahorian}@binghamton.edu

ABSTRACT

In this paper, we evaluate the front-end of Automatic Speech Recognition (ASR) systems, with respect to different types of spectral processing methods that are extensively used. Experimentally, we show that direct use of FFT spectral values is just as effective as using either Mel or Gammatone filter banks, as an intermediate processing stage, if the cosine basis vectors used for dimensionality reduction are appropriately modified. Furthermore it is shown that trajectory features computed over intervals of approximately 300ms are considerably more effective, in terms of ASR accuracy, than are delta and delta-delta terms often used for ASR. Although there is no major performance disadvantage if a filter bank is used, simplicity of analysis is a reason to eliminate this step in speech processing. The experimental results which confirm the above assertions are based on the TIMIT phonetically labeled database. The assertions hold for both clean and noisy speech.

Index Terms: DCTC/DCSC, MFCC, Gammatone filter bank, Mel filter bank, ASR.

1. INTRODUCTION

All automatic speech recognizers perform spectral analysis at the front end which converts the speech signal, possibly noisy and/or degraded, into values from which useful features can be easily computed. The front end spectral analysis is performed by calculating the short time Fourier transform (STFT) of the speech signal, either using an FFT, a filter bank, or a combination of the two methods. For the combination method, the filter bank is approximated by summing weighted combinations of FFT magnitude values. The filter bank approach, even if derived from FFT values, is thought to be advantageous since it can be designed to mimic the functionality of the cochlea of the human auditory system, such as a nonlinear ("warped") frequency scale.

The majority of ASR systems are implemented using a Mel filter bank as the spectral analysis front end, followed by a cosine transform based feature extraction which is shown to outperform other signal processing methods [1]. Very recently, another filter bank has been presented as a superior alternative to the triangular-shaped Mel filters called the Gammatone filter bank, which simulates the motion of the basilar membrane within the cochlea of the human auditory system. It was first introduced by Johannsma (1972) to describe the shape of the impulse response function of the auditory system as estimated by the reverse correlation function of neural firing times. The general thinking is that since the Gammatone filter bank approximates the human auditory system better than the Mel filter bank, it should also be superior for ASR applications [2].

The Gammatone filter is defined in the time domain (impulse response function) as:

$$g(t) = at^{n-1}e^{-2\pi bt}\cos(2\pi ft + \emptyset) \quad (1)$$

where f is the frequency, $\emptyset$ is the phase of the carrier, a is the amplitude, n is the filter order, b is the bandwidth and t is time.

Front-end spectral analysis can also be performed without using any filter bank, but simply using an FFT directly. In either case, spectral values (that is FFT values or filter bank outputs, both converted to magnitudes), are typically reduced in dimensionality using some type of cosine transform. If the filter bank step is used, cosine basis vectors can be used directly. However, if the FFT magnitudes are used as the direct input to the cosine transform, the cosine basis vectors should be modified to account for the non-uniform frequency resolution. In order to incorporate spectral trajectory information into ASR feature sets, additional terms are generally computed from blocks of frame-based features, such as delta terms.

In the following sections we compare spectral features computed as cosine transforms of filter bank outputs with features computed as modified cosine transforms (DCTCs) of FFT spectral magnitudes directly. We also compare delta type trajectory features with trajectory features computed over much longer time intervals using another set of modified cosine basis vectors (DCSCs). More details of the more common spectral and feature calculation method (MFCCs with delta and delta-delta terms are given in [3] and [4]. More details of the DCTC/DCSC general method are given in [5], [6] and [12]. All the methods are evaluated using as much similarity of parameters and recognizer as feasible (such as frequency range, # of HMM mixtures, etc.) in order to make comparisons most meaningful.

2. FFT BASED SPECTRAL ANALYSIS

The most common spectral analysis method for speech recognition uses a frame-based approach in which the time varying speech signal is described by a stream of feature vectors, with each vector reflecting the spectral magnitude properties of a relatively short (10-30ms) segment (frame) of the signal. For experimental results reported in this paper, 16 kHz sampling rate speech signals are short-time Fourier transform (STFT) analyzed using a 10ms Kaiser window with a frame space of 2ms. The spectrogram of a typical speech signal is as shown in Figure 1. The FFT spectral values are used as the front-end for DCTC/DCSC feature extraction, as described later. The frame length and frame spacing mentioned were empirically determined as providing most accurate ASR results.

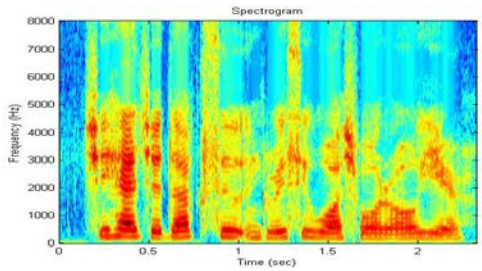

Figure 1: FFT spectrogram

3. FILTER BANK BASED SPECTRAL ANALYSIS

A filter bank can be regarded as a crude model for the initial stages of transduction in the human auditory system. A set of band pass filters is designed so that a desired portion of the speech band is entirely covered by the combined pass bands of the filters composing the filter bank. The output of the band pass filters are considered to be the time varying spectral representation of the speech signal.

For the experiments given in this paper, we evaluate two commonly used filter banks: the Mel filter bank and Gammatone filter bank. Either the DCTC/DCSC method (but without frequency warping) or the more common method used for MFCC features (i.e., delta terms rather than DCSCs) are used. Results are compared for the filter bank approaches versus the FFT-only spectral method.

3.1. MEL FILTER BANK

The Mel filter bank is a series of triangular band pass filters, as depicted in Figure 2, designed to simulate the band pass filtering believed to be similar to that occurring in the auditory system.

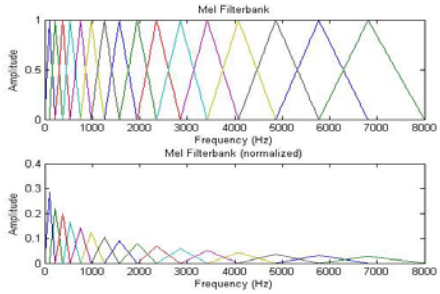

Figure 2: Frequency response of 16 channel Mel filter bank and the normalized versions of the filters, as used for MFCCs.

To convert the frequency in Hz into frequency in Mels the following equation is used:

$$m = 1127.01048 * log_e\left(1 + \frac{f}{700}\right) \quad (2)$$

On a linear frequency scale, the filter spacing is approximately linear up to 1000 Hz and approximately logarithmic at higher frequencies. For actual implementation, the Mel filter bank is computed by first computing the power spectrum with an FFT, and then multiplying the power spectrum by the Mel filter bank coefficients. In Figure 3 is shown a spectrogram based on 32 Mel filters. Note that this spectrogram is qualitatively similar to the direct FFT spectrogram shown in Figure 1. The details of the two spectrograms are quite different since the frequency range is more quantized in Figure 3 and the frequency scale is effectively in Mels rather than linear. However, it should be noted that the Mel spectrogram, or Mel filters, are derived from the FFT spectral values and thus are simply an intermediate step in processing.

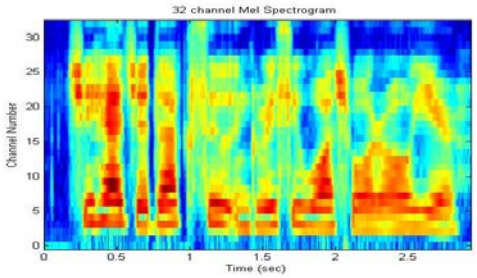

Figure 3: 32 channel Mel spectrogram

3.2. GAMMATONE FILTER BANK

A Gammatone filter is a linear filter with impulse response described as the product of a (gamma) distribution and sinusoidal (tone), hence the name Gammatone. The filter bank is a combination of individual Gammatone filters with varying bandwidth based on the Equivalent Rectangular Bandwidth (ERB) scale. For moderate sound pressure levels, Moore et al [7] [8] estimated the size of ERBs for humans as:

$$ERB[f] = 24.7 + 0.108 * f_c \quad (3)$$

The value ERB[f] is used as the unit of center frequency f_c on the ERB scale. For example, the value of ERB[f] for a center frequency of 1 kHz is about 132.64, so an increase in frequency from 975 to 985 Hz represents a step of one ERB[f]. Each step in ERB roughly corresponds to a constant distance of about 0.89 mm on the basilar membrane [9].

As the center frequency increases the bandwidth of the filter bank increases. A 16 channel Gammatone FFT based filter bank frequency response is shown in Figure 4.

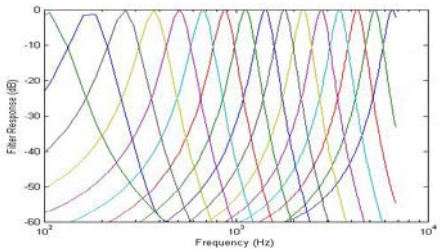

Figure 4: Frequency response of 16 channel Gammatone filter bank

The Gammatone filter bank can be implemented using sums of weighted FFT power spectrum values [10], exactly as for the Mel filter bank except using the weights corresponding to Figure 4, rather than the Mel filter weights shown in Figure 2. Alternatively, the Gammatone real filters can be implemented as actual IIR or FIR filters, followed by rectification and low pass filters, as depicted in Figure 5. Figure 6 depicts the Gammatone spectrogram of the same sentence as was used to construct the spectrograms for Figures 1 and 3.

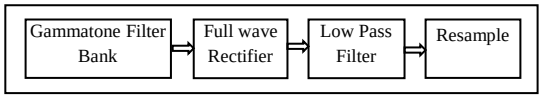

Figure 5: Block diagram of Gammatone using actual filters (difference equations) in first block

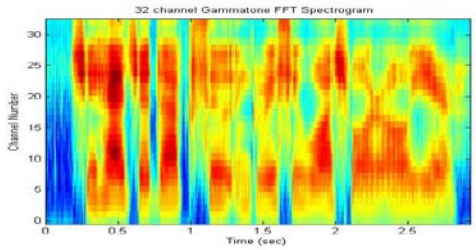

Figure 6: 32 channel Gammatone spectrogram

4. DCTCs/ DCSCs BASED FEATURE EXTRACTION

Typically FFT spectral magnitudes or filter bank outputs are dimensionality reduced with a cosine or cosine-like transform for each frame of spectral values. Several frames of cosine transform coefficients are further processed in overlapping sliding blocks to form spectral trajectory features. Although both of these steps are very "standard," especially for the case of Mel filter bank spectral values for the preceding step, in this section we review these transforms especially as they relate to using FFT spectral values directly.

The first step of this feature calculation is to compute DCTC terms from the spectrum X, with the frequency f normalized to a [0, 1] range, as follows

$$DCTC(i) = \int_0^1 a\left(X\left(g(f)\right)\right)\phi_i(f)df \qquad (4)$$

In this equation, i is the DCTC index, $a(X)$ is a nonlinear amplitude scaling and $g(f)$ a nonlinear frequency warping. $\Phi_i(f)$ is the i^{th} basis vector over frequency computed as:

$$\phi_i(f) = \cos[\pi i g(f)]\frac{dg}{df} \qquad (5)$$

The crucial elements of this approach are the selection of the nonlinear amplitude scaling $a(X)$ and the nonlinear frequency scaling $g(f)$ so that the cosine transform is with respect to a perceptual scale. In practice, the scaling $a(X)$ is typically a log, and the scaling $g(f)$ is a Mel-like function unless the first step is a Mel-like filter bank, in which case g(f) = f, dg/df = 1, and the basis vectors are "regular" cosines.

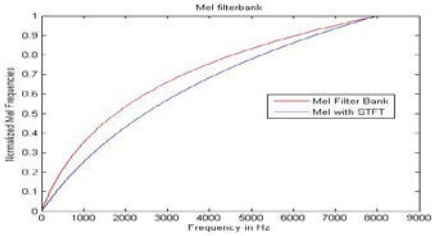

Figure 7: Mel frequency warping used for Mel filter bank center frequencies (top red curve), and "optimum" Mel frequency warping used for FFT-only/DCTC/DCSC method (bottom blue curve)

For the case of FFT-only spectral analysis frequency, g(f) is a Mel-like "warping" function, which has the effect of modifying the cosine basis vectors, according to Eq. 5. The results presented in this paper for the DCTC/DCSC expansion of FFT spectra were based on this Mel-like warping (lower blue curve in Figure 7), which was empirically found to perform better than the more precise Mel warping as given in Eq. 2 and also depicted in Figure 7.

In order to create the DCSC features that represent the spectral evolution of DCTCs over time, as an alternative to delta and delta-delta terms typically used with MFCCs, a cosine basis vector expansion over time is performed using overlapping blocks of DCTCs. That is, the DCSCs are computed as:

$$DCSC(i,j) = \int_0^1 DCTC\big(i,h(t)\big)\Theta_j(t)dt \qquad (6)$$

where $\Theta_j(t)$ is the j^{th} basis vector over time computed as:

$$\Theta_j(t) = \cos[\pi i h(t)]\frac{df}{dt} \qquad (7)$$

In this equation, $h(t)$ is a time warping function and t is normalized to [0,1] over a selected segment (a "block"). In practice, t is discrete, corresponding to a frame index, and the integral is computed using a sum of all frames in the block. The calculation is repeated for each overlapping block, with the block spacing some integer multiple of the frame spacing.

5. PHONETIC RECOGNITION EXPERIMENTS

Phonetic recognition experiments were conducted using the TIMIT phonetically-labeled database. 3296 sentences from 462 speakers were used for training and 1344 sentences from 168 speakers were used for test. SA sentences were excluded. A frequency range of 100 to 8000 Hz was used for all experiments. Experiments were conducted with clean, 20 dB SNR, 10 dB SNR, and 0dB SNR speech. For all conditions, training and test conditions were matched with respect to noise; additive white Gaussian noise was used for noise.

The objective of the experiments was to compare phoneme recognition accuracy for four spectral analysis methods, as depicted in Figure 8, and also to compare to a control case (13 MFCCs with delta and acceleration terms, or 39 total terms, derived from a Mel filter bank, as implemented in HTK).

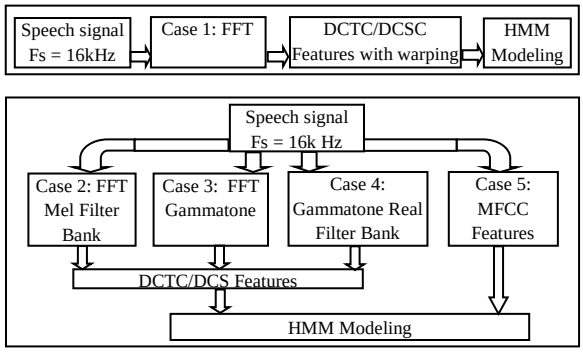

Figure 8: Block diagram of phonetic speech recognition process

Five cases, as depicted in Figure 8, and outlined below were tested.

Case 1: FFT spectrum directly used as front end for DCTC/DCSC feature, using frequency warping (Figure 7).

Case 2: DCTC/DCSC feature extraction applied to Mel filter bank spectrum. Since the filter bank already has warping in it, the DCTC basis vectors have no warping.

Case 3&4: Gammatone filter banks (FFT-based and actual filters cases) used as front end for DCTC/DCSC features, with no frequency warping used for DCTCs.

Case 5: HTK MFCC features with delta terms.

For all experiments with DCTC/DCSC features, a frame spacing of 2ms (500 frames per second) was used. Blocks were comprised of 150 frames (300ms) and spaced 8ms apart (125 blocks per second). Experiments were conducted with both 78 features (13 DCTCs times 6 DCSCs), and the more standard 39 features (13 DCTCs times 3 DCSCs).

HMMs with 3 hidden states from left to right with 16 Gaussian mixtures were used for phonetic recognition experiments. A total of 48 (eventually reduced to 39 phones) context independent monophone HMMs were created using the HTK toolbox (Ver3.4) [12]. The bigram phone information extracted from the training data was used as the language model.

6. RESULTS

Phonetic recognition accuracy (based on 39 phones) obtained for all 5 cases is given in Table 1. It can be seen that there is negligible or no improvement when filter bank techniques are used. For results in Table 1, 39 features were used. The experiment was repeated with 78 features for all cases except MFCC, and results are given in Table 2.

Table 1: Accuracy (%) comparison for 39 features

SNR (dB)	FFT only	Mel FB	Gammatone FFT FB	Gammatone Real FB	MFCC
Clean	69.2	68.5	69.8	69.1	62.8
20 dB	64.2	63.5	63.7	63.4	
10 dB	56.3	55.0	55.8	55.0	
0 dB	42.2	41.5	41.4	40.5	

Table 2: Accuracy (%) comparison for 78 features

SNR (dB)	FFT only	Mel FB	Gammatone FFT FB	Gammatone Real FB
Clean	71.2	69.7	71.1	70.1
20 dB	65.8	64.7	65.8	64.9
10 dB	58.0	58.1	58.1	56.9
0 dB	43.4	42.5	42.8	41.8

Both case 2 and case 5 in Table 1 used Mel warping, but there is a considerable difference in the performance of the two. To investigate the possible reason for this, the delta terms and the DCSC terms were removed from MFCC using HTK and Mel filter bank respectively, and the results shown in Table 3 were obtained.

Table 3: Performance comparison of MFCC and Mel filter bank.

# Channels	Mel FB		MFCC{HTK}	
	100-6000	100-8000	100-6000	100-8000
32 {FL=10ms, FS =2ms}	53.9	53.9	52.8	53.2
32 {FL=25ms, FS =10ms}	49.1	49.1	50.7	50.2
20 {FL=10ms, FS =2ms}	53.3	53.3	53.5	53.4
20 {FL=25ms, FS =5ms}	48.9	48.9	50.6	50.6
26 {FL=10ms, FS =2ms}	53.9	53.9	51.0	50.5
26 {FL=25ms, FS =10ms}	50.3	50.3	53.9	50.4

'FL' is the frame length and 'FS' is the frame spacing that is used. The results show that when the delta terms and the DCSC terms are removed, the performance of MFCC computed using HTK is similar to that of the Mel filter bank implemented in our code. Thus, presumably, the advantage of our Mel filter bank versus the HTK filter bank is due to the difference in the way the spectral change information was represented. Mel FB and MFCC (HTK) was also tested by varying the high frequency range from 6000- 8000 Hz and both the cases are reported in Table3.

As yet another test, Table 4 shows the accuracy obtained with the Gammatone filter bank as the number of channels is varied from 8 to 128. Although there is a very slight improvement when using 64 channels, this comes at the expense of more computational time and complexity, so we considered the "standard" as 32 channels for the Gammatone filters, and used 32 channels for all the results (except for Table 4 results) in this paper.

Table 4: FFT Gammatone performance as number of filters is varied.

SNR (dB)/Channels	8	16	32	48	64	128
Clean	64.5	69.4	71.7	71.3	71.4	71.1
20 dB	60.1	64.8	65.8	65.8	65.9	65.9
10 dB	50.8	56.0	58.1	58.1	59.3	58.1

To test the statistical significance of the differences in accuracy for the results given in this paper, we performed several we performed several t-tests by dividing the 1344 sentences of test into sets of 96 sentences each. Using the means and variances of the groups of 14 independent tests, and using standard statistical hypothesis testing methods [13], it was determined that 2% differences are significant at the 97.5% confidence level, and 1% differences are significant at the 90% confidence level. Thus, for example, in Table 1, for a fixed SNR, many of the results are statistically similar, except for MFCC results, which are lower than for all the other methods shown.

7. CONCLUSIONS

From the experimental data, we conclude that FFT-based spectral analysis in both clean and noisy conditions with a Mel-like frequency scale incorporated using frequency warping for DCTC features performs nearly identically to cochlea-motivated filter bank spectral analysis. Directly using the FFT spectrum, without the intermediate filter bank prior to feature calculations, has the advantage of simplicity and would appear to be a better front end strategy for spectral front end calculations for speech processing. The DCSC method for computing spectral trajectory features is experimentally shown to result in much higher ASR accuracy than obtained with delta and delta-delta terms.

8. ACKNOWLEDGEMENTS

This material is based on research sponsored by the Air Force Research Laboratory under agreement number FA87501210093. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Laboratory or the U.S. Government.

9. REFERENCES

[1] S. B. D and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *IEEE Trans. Acoustic., Speech, Signal Processing*, vol. ASSP- 28, no. 4, pp. 357-366, 1980

[2] Yuxuan Wang, Kun Han, DeLiang Wang "Exploring Monaural Features for Classification-Based Speech Segregation," *IEEE transactions on audio, speech and language processing, Vol. 21, No. 2, February 201.*

[3] Md. Afzal Hossan, S. Memon, M A Gregory, "A novel approach of MFCC feature extraction," *IEEE Trans. On Signal Processing and Communication 2010 4th international conference.*

[4] Wu Junqin, Yu Junjun, "An Improved Arithmetic of MFCC in Speech Recognition System," *IEEE 201, pp 719-722*

[5] S.A. Zahorian, Silsbee, P., and Wang, X., "Phone Classification with Segmental Features and a Binary-Pair partitioned Neural Network Classifier," *Proc. ICASSP* 1997, pp.1011-1014, 1997.

[6] M. Karjanadecha and S.A. Zahorian, "Signal Modeling for High-Performance Isolated Word Recognition," *IEEE Trans. on Speech and Audio Processing*, 9(6), pp.647-654, 2001.

[7] S. Strahl, "Analysis and design of Gammatone signal models," *J. Acoust. Soc. Am.* 126, pp. 2379-2389, 2009.

[8] B. Moore, R. Peters, and B. Glasberg, "Auditory filter shapes at low center frequencies," *J. Acoust. Soc. Am*. 88, 132–140, 1990.

[9] B. Moore and B. Glasberg, "A revision of Zwicker's loudness model," *Acta. Acust. Acust*. 82, 335–345, 1996

[10] Holdsworth J. et al. "Implementing a Gamma Tone Filter Bank," in SVOS Final Report – *Part A: The Auditory Filter bank, MRC Applied Psychology Unit*, Cambridge, England, 1988.

[11] L. Rabiner, B.H. Juang, "Fundamentals of speech Recognition," *Prentice Hall Signal Processing Series*, 1993.

[12] S.A. Zahorian, Hongbing Hu, Zhengqing Chen, Jiang Wu, "Spectral and Temporal Modulation Features for Phonetic Recognition," *Interspeech* 2009.

[13] Will Thalheimer, Samantha Cook, "How to calculate effect sizes from published research: A simplified methodology,"*A Work-Learning Research Publication, Published August 2002.*

A5 H. Hu, P. Guzewich, and S. A. Zahorian, "A Further Comparison of Fundamental Frequency Tracking Algorithms," *166th Meeting of the Acoustical Society of America*, San Francisco, Vol. 134, No. 5, Pt. 2, pp. 4068, Dec 2–6, 2013.

"Yet another Algorithm for Pitch Tracking -YAAPT" was published in a 2008 JASA paper (Zahorian and Hu), with additional experimental results presented at the fall 2012 ASA meeting in Kansas City. The results presented in both the journal paper and at the fall 2012 meeting indicated that YAAPT generally has lower error rates than other widely used pitch trackers (YIN, PRAAT, RAPT). However, even YAAPT-created pitch tracks had significant "large" errors (pitch doubling and pitch-halving) for both clean and noisy speech. Recently additional post-processing heuristics have been incorporated to reduce the incidence of these type errors—thus reducing the need for hand correcting pitch tracks for situations where extremely accurate tracks are desired. For the case of an all-voiced track, interpolation through unvoiced intervals has been improved. The updated version of YAAPT is presented along with experimental results. The experiments are conducted with multiple databases, including British English, American English, and Mandarin Chinese. For most conditions evaluated, YAAPT gives better performance than the other fundamental frequency trackers.

Words in abstract: 167

Technical area: Speech Processing and Communication Systems

(PACS) Subject classification numbers(s): 43.72.Ar, 43.72.Ne

No preference for lecture versus poster

LIST OF SYMBOLS, ABBREVIATIONS AND ACRONYMS

ARPA	Advanced Research Project Agency
AFRL	Air Force Research Laboratory
ASR	Automatic Speech Recognition
DCT	Discrete Cosine Transform
DCTC	Discrete Cosine Transform Coefficient
DCS	Discrete Cosine Series
DCSC	Discrete Cosine Series Coefficient
FFT	Fast Fourier Transform
HMM	Hidden Markov Model
HTK	Hidden Markov Model Toolkit
LDC	Linguistic Data Consortium
LM	Language Model
MFCC	Mel-Frequency Cepstral Coefficient
NN	Neural Network
OOV	Out of Vocabulary
PNCC	Power Normalized Cepstral Coefficient
RASC863	Regional Accented Speech Corpus by National 863 Project
863	Mandarin Chinese Database by National 863 Project
TIMIT	Texas Instruments--Massachusetts Institute of Technology
YAAPT	Yet Another Algorithm for Pitch Tracking